

INFORMÁTICA II
(2025)

GUÍA PARA ESTUDIANTES 2016

- INGENIERÍA EN ELECTRÓNICA

DEPARTAMENTO DE CIENCIAS APLICADAS Y TECNOLOGÍA


UNIVERSIDAD
NACIONAL
DE MORENO

DEPARTAMENTO DE ECONOMÍA Y ADMINISTRACIÓN

Director General - Decano
Pablo A. TAVILLA

Licenciatura en Relaciones del Trabajo
Coordinadora - Vicedecana
Sandra M. PÉREZ

Licenciatura en Administración
Coordinador - Vicedecano
Pablo A. TAVILLA (a cargo)

Licenciatura en Economía
Coordinador - Vicedecano
Alejandro L. ROBBA

Contador Público Nacional
Coordinador - Vicedecano
Alejandro A. OTERO

Departamento de Economía y Administración
(0237) 466-7186/1529/4530
(0237) 462-8629
(0237) 460-1309
Interno 124
Oficina A101
eya@unm.edu.ar

DEPARTAMENTO DE HUMANIDADES Y CIENCIAS SOCIALES

Directora General - Decana
M. Patricia JORGE (ad honórem)

Licenciatura en Trabajo Sicial
Coordinadora - Vicedecana
M. Claudia BELZITI

Licenciatura en Comunicación Social
Coordinador - Vicedecano
Roberto C. MARAFIOTI

Licenciatura en Educación Secundaria
Coordinadora - Vicedecana
Lucía ROMERO

Licenciatura en Educación Inicial
Coordinadora - Vicedecana
Nancy B. MATEOS

Departamento de Humanidades y Ciencias Sociales
(0237) 466-7186/1529/4530
(0237) 462-8629
(0237) 460-1309
Interno 125
Oficina A 104
hycs@unm.edu.ar

DEPARTAMENTO DE CIENCIAS APLICADAS Y TECNOLOGÍA

Director General - Decano
Jorge L. ETCHARRÁN

Ingeniería en Electrónica

Licenciatura en Gestión Ambiental
Coordinador - Vicedecano
Jorge L. ETCHARRÁN (ad honórem)

Licenciatura en Biotecnología
Coordinadora - Vicedecana
Marcela A. ALVAREZ (int.)

Departamento de Ciencias Aplicadas y Tecnología
(0237) 466-7186/1529/4530
(0237) 462-8629
(0237) 460-1309
Interno 129
Oficina B 203
cayt@unm.edu.ar

DEPARTAMENTO DE ARQUITECTURA, DISEÑO Y URBANISMO

Directora General - Decana
N. Elena TABER (a cargo)

Arquitectura
Coordinadora - Vicedecana
N. Elena TABER (int)

Departamento de Arquitectura, Diseño y Urbanismo
(0237) 466-7186/1529/4530
(0237) 462-8629
(0237) 460-1309
Interno 428
Oficina E 102
adyu@unm.edu.ar

INFORMÁTICA II

(2025)

GUÍA PARA ESTUDIANTES

2016



Colección: Cuadernos de Cátedra
Directora: Adriana M. del H. Sánchez
Autor: Osvaldo Pini

Pini, Osvaldo Mario
Informática II, 2025 : guía para estudiantes / Osvaldo Mario Pini. - 1a ed . - Moreno : UNM Editora, 2016.
Libro digital, PDF - (Cuadernos de cátedra)

Archivo Digital: descarga y online
ISBN 978-987-3700-24-8

1. Informática. I. Título.
CDD 005.3

1.ª edición digital: marzo de 2016

Av. Bartolomé Mitre N° 1891, Moreno (B1744OHC), prov. de Buenos Aires, Argentina
(+54 237) 466-7186/1529/4530
(+54 237) 462-8629
(+54 237) 460-1309
Interno: 154
unmeditora@unm.edu.ar
http://www.unm.edu.ar/editora

MATERIAL DE DISTRIBUCIÓN GRATUITA







INTRODUCCIÓN

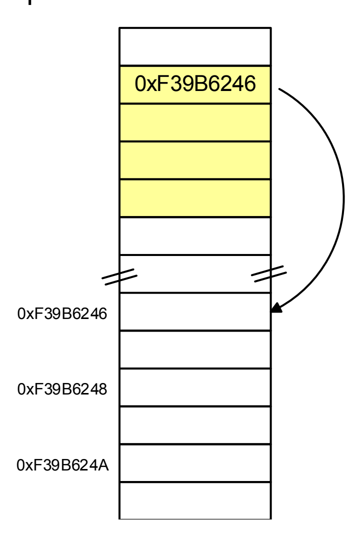
Informática II (2025) es la continuación de Informática I y presenta, como es lógico de esperar, mayor complejidad. Los presupuestos básicos de la programación en lenguaje C ya estudiados en el curso anterior serán el punto de partida para el recorrido que propone este cuadernillo, cuyos temas abarcan toda la asignatura y sobre los que haremos a continuación una suscinta descripción.
El primer tema, “Punteros”, desarrolla el tipo de datos que se utiliza para acceder a las posiciones de memoria en un sistema de procesamiento. Esto permite modificar el contenido de la memoria en el momento que se lo desee y pasar parámetros por referencia a una función.
El segundo tema, “Recursividad”, aborda un manejo particular de las funciones: las que se invocan a sí mismas.
En “Uniones” veremos un tipo de datos que comparten el uso de la memoria. En una unión, todos los datos comienzan en la misma posición de memoria.
“Campos de bits”, el cuarto tema, desarrolla una clase de datos que permite seleccionar el tamaño de las variables en cantidad de bits. Según lo disponga el programador, se puede seleccionar una variable para que tenga un largo de un bit o incluso más.
El quinto tema se centra en los operadores a nivel de bits, que son operadores lógicos que permiten cambiar el estado de los bits seleccionados de un determinado dato.
Abordaremos luego el tema de los archivos, que permiten almacenar los datos procesados en un disco, enviarlos a una impresora o comunicar los datos a otro sistema.
En “Variables dinámicas” estudiaremos una zona de la memoria, llamada dinámica, que está disponible cuando se ejecuta un programa y se le solicita al sistema operativo que le conceda la utilización de esta memoria.
Por ultimo, abordaremos el estudio de estructuras de datos que requieren un manejo especial, ya que siguen reglas particulares para su acceso. Por un lado, tenemos las de lectura destructiva, llamadas colas, colas circulares y pilas. Por el otro, la lista, que es una estructura que cambia su tamaño a medida que se van agregando o eliminando datos.
Todos estos temas se desarrollan conforme con una progresiva complejización que el alumno deberá procesar y trabajar durante la cursada, y contará con el apoyo y refuerzos necesarios por parte del cuerpo docente de la materia.

Ing. Osvaldo M. Pini
Informática II (2025)
Departamento de Ciencias Aplicadas y Técnología

TEMA 1

PUNTEROS

Un **puntero** es un tipo de dato que permite almacenar una dirección de memoria.
Esto quiere decir que un puntero apunta a una determinada ubicación en la memoria.



Declaración

Para poder declarar una variable de tipo puntero se necesita asignarle un nombre, precedido por un tipo asociado y el operador de indirección (*).”Tipo asociado” es cualquier tipo de dato, tanto los definidos por el compilador como los definidos por el usuario.

Sintaxis

tipo asociado* nombre puntero;

```
int * punt;
```

En este ejemplo se declara a la variable *punt* como un puntero que está asociado al tipo de dato int, es decir que *punt* es un puntero a entero.

Tamaño de los punteros

Dado que un puntero puede almacenar una dirección de memoria, su tamaño será el necesario para guardar en forma adecuada dicho valor.
En la actualidad, este valor es de 4 bytes (32 bits), lo que permite direccionar memorias de 4GBy. Probablemente en un futuro su tamaño puede cambiar; no obstante siempre se puede obtener su dimensión mediante el uso del comando sizeof.

Operadores

Los punteros tienen asociados dos operadores: el operador de direccionamiento (&) y el operador de indireccionamiento (*).

Operador de direccionamiento (&)

Este operador permite poder obtener la dirección de una variable.

```
int a;
int *pi;
pi=&a;
```

En este caso, el puntero pi guardará la dirección de comienzo de la variable **a**.

Ejemplo

```
//------------------------------------------------------------------
#include<stdio.h>
//------------------------------------------------------------------
int main(void)
{
    int i,*pi;
    printf("\nLa dirección de i es :%p",&i);
    /* Se muestra la dirección de i utilizando el operador & */
    pi=&i;
    printf("\nLa dirección de i es :%p\n\n",pi);
    /* Indica la dirección utilizando un puntero */
    return 0;
}
//------------------------------------------------------------------
```

Se puede observar cómo se obtiene la dirección de una variable utilizando tanto el operador de direccionamiento como un puntero.
Se observa que para mostrar el contenido de un puntero mediante el uso de la función *printf*, se debe utilizar el especificador de formato %p.

Ejemplo

```
//------------------------------------------------------------------
#include<stdio.h>
//------------------------------------------------------------------
int main(void)
{
    int i=1,*pi=&i;
    /* Declara e inicializa al puntero pi con la dirección de la
      variable i */
    printf(" i = %d\n",i);
    printf("dir i = %p\n",&i);
    /* Muestra la dirección de i utilizando el operador & */
    printf(" i = %d\n",*pi);
    printf("dir i = %p\n",pi);
```

```
    /* Muestra la dirección y el contenido de i utilizando el
      puntero pi */
    return 0;
}
//------------------------------------------------------------------
```

En este programa se muestra cómo un puntero apunta a una variable y cómo se puede acceder al valor almacenado en ella utilizando un puntero.

Operador de indirección (*)

Este operador permite acceder al contenido de la memoria que es apuntada por el puntero.

```
int a,*pi;
pi=&a;
*pi=4;
```

Vemos que el puntero **pi** apunta a la variable **a,** Al acceder de este modo al contenido de la memoria, se escribió en esa dirección el valor 4, modificando indirectamente el valor de la variable **a**.

Ejemplo

```
//------------------------------------------------------------------
#include<stdio.h>
//------------------------------------------------------------------
int main(void)
{
    int i,*pi,*pj;
    char c,*pc;
    float f,*pf;
    pi=&i;
    pc=&c;
    pf=&f;
    /* Inicializa a los punteros con las direcciones de las
      variables correspondientes */
    printf("\nLa dirección de i es: %p\n",&i);
    printf("El contenido de pi es: %p\n",pi);
    printf("\nLa dirección de c es: %p\n",&c);
    printf("El contenido de pc es: %p\n",pc);
    printf("\nLa dirección de f es: %p\n",&f);
    printf("El contenido de pf es: %p\n",pf);
    /* Muestra las direcciones de las variables y sus contenidos
      utilizando los punteros */
    pj=pi;
    printf("\nLa dirección de i es: %p\n",&i);
    printf("El contenido de pi es: %p\n",pi);
    printf("El contenido de pj es: %p\n",pj);
    pc=0x7A34;
    pf=567;
    /* Le asigna a los punteros direcciones de memoria */
    printf("\nEl contenido de pc es: %p\n",pc);
    printf("El contenido de pf es: %p\n",pf);
```

```
    /* Muestra los valores almacenados por los punteros */
    return 0;
}
//-------------------------------------------------------------------
```

Aquí comprobamos cómo se pueden apuntar los punteros a diferentes variables y posiciones de memoria.

Ejemplo

```
//-------------------------------------------------------------------
#include<stdio.h>
//-------------------------------------------------------------------
int main(void)
{
    int i,*pi;
    float j,*pj;
    char k,*pk;
    i=1000;
    pi=&i;
    j=3.18;
    pj=&j;
    k='c';
    pk=&k;
    /* Inicializa a las variables y a los punteros */
    printf("\n%d\t%p\t%f\t%p\t%c\t%p\n",i,pi,j,pj,k,pk);
    /* Muestra los contenidos de las variables y los punteros */
    for(i=-5;i<6;i++,pi++,pj++,pk++)
{
        /* Incrementa a los punteros y muestra el estado de las
          variables, los punteros y los contenidos de las
          direcciones de memoria a las que apuntan */
        printf("\n%d\t%p\t%f\t%p\t%c\t%p",i,pi,j,pj,k,pk);
        printf("\n%d\t%p\t%f\t%p",*pi,pi,*pj,pj);
        printf("\t%c\t%p\n\n",*pk,pk);
    }
    printf("\n\n");
    return 0;
}
//-------------------------------------------------------------------
```

En este ejemplo observamos cómo se van incrementando los punteros y permiten recorrer la memoria y ver su contenido.

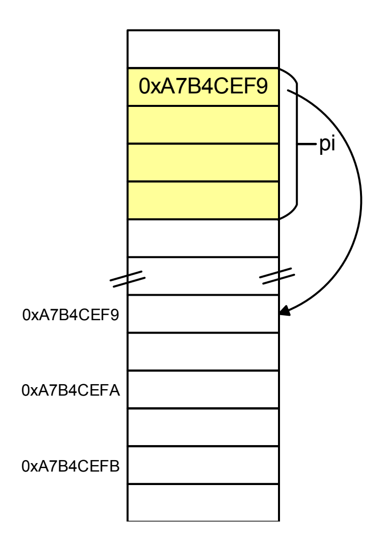
Operaciones que admiten los punteros

Asignación

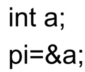
A un puntero se le puede asignar un valor entero: el significado es una dirección de memoria. Debemos recordar que la memoria está dividida en celdas de 1 byte cada una, por lo que, entre una posición y la siguiente, la distancia que los separa es uno (1). Esto implica que,no

se puede acceder a una posición intermedia en una celda.
El operador que se utiliza es el de **asignación** (**=**).

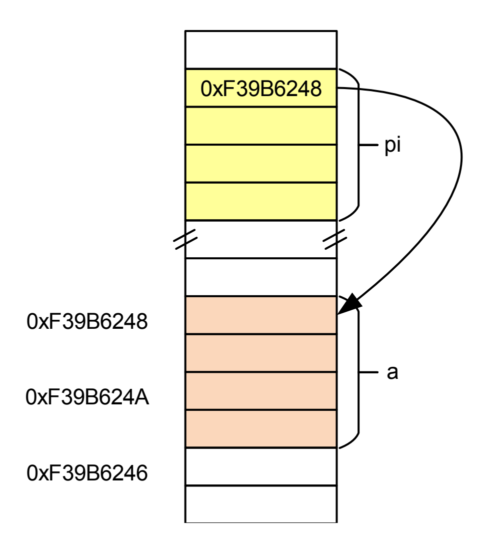
```
int *pi;
pi=0xA7B4CEF9;
```



En este caso se le asigna un valor numérico entero.
También admiten la asignación de la dirección de una variable. Para realizar esta operación se utiliza el operador de direccionamiento **(&)**.



```
int a;
pi=&a;
```



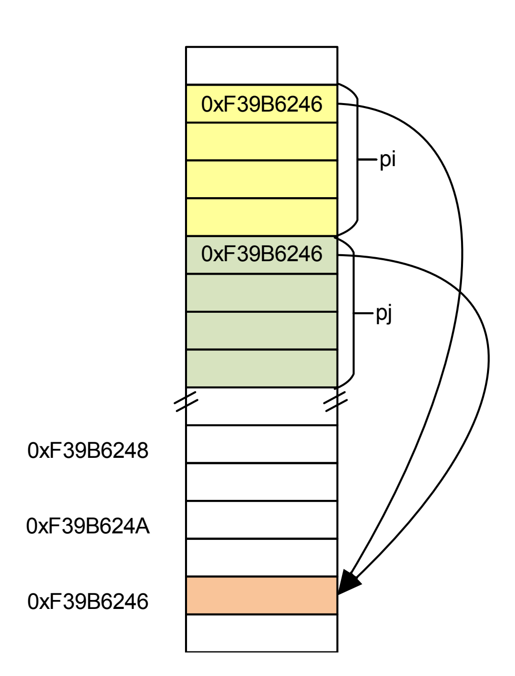
Ejemplo

```
//--------------------------------------------------------------------
#include<stdio.h>
//--------------------------------------------------------------------
int main(void)
{
    int i,*pi;
    pi=&i;
    *pi=45;
    /* Se modifica el contenido de la variable a través del
      puntero */
    printf("\nEl valor que contiene i es: %d",i);
    i=39;
    printf("\nEl valor que contiene i es: %d\n",*pi);
    /* Se muestra el contenido de la variable utilizando el
```

```
        puntero */
    return 0;
}
//-------------------------------------------------------------------
```

Se observa en el ejemplo cómo se puede acceder al contenido de una variable utilizando un puntero, ya sea para escribir en la variable como para leerla.

Otro valor que se le puede asignar a un puntero es otro puntero. Como es general en C, a toda variable de un determinado tipo se le puede asignar otra variable del mismo tipo.




```
int *pj,*pi;
pj=0xF39B6246;
pj=pi;
```

Hay que tener en cuenta que cuando se declara un puntero hay que inicializarlo, ya que el valor que contiene luego de la declaración es basura. En este caso se dice que el puntero está descontrolado, y pueden cometerse errores muy peligrosos en el caso de utilizarlo.

Aritméticas

Los punteros admiten las operaciones de suma y resta. Estas operaciones se realizan utilizando los operadores de adición (+) y sustracción (-).
Los valores que se adiconan o sustraen pueden ser enteros u otro puntero.
Si se le adiciona un entero, se va a desplazar hacia posiciones superiores de la memoria. En cambio, si se le sustrae un entero, este se desplazará hacia posiciones inferiores. La cantidad de bytes que se desplaza el puntero es el tamaño del tipo asociado. Es decir, si el tipo asociado al puntero es un int, este se desplazará 4 bytes por cada unidad que se le adicione o sustraiga.

```
int *pi;
char *pc;
short int *ps;
pi=0xb3500a00;
pc=0xb3500a00;
```

```
ps=0xb3500a00;
pi+=3;
pc+=3;
ps+=3;
```

En este caso el puntero pi se desplazará 12 bytes, el puntero pc cambiará en 3 bytes y el puntero ps se trasladará 6 bytes, todos desde la posición inicial de cada uno.
El valor final que contendrán los punteros será: `pi->0xb3500a0c`,`pc->0xb3500a03` y `ps->0xb3500a06`.
Los valores numéricos que se le asignan están, por lo general, en formato hexadecimal. No hay ningún impedimento para que se utilice otro formato, pero como las direcciones de memoria se identifican en hexadecimal, este formato es el que mejor se adapta para este uso.
También aceptan las operaciones de incremento o decremento, ya que sería lo mismo que sumarle o restarle la unidad a un puntero.

Ejemplo

```
//-----------------------------------------------------------------------
#include<stdio.h>
//-----------------------------------------------------------------------
int main(void)
{
    short int *pi,i;
    char *pc;
    float *pf;
    pi=pc=pf=0x02000000;
    /* Inicializa a los punteros con una determinada dirección */
    for(i=0;i<10;i++)
    {
        ‘/* Muestra el estado de los punteros y los incrementa */
        printf(“%p\t%p\t%p\n”,pi,pf,pc);
        pi++;
        pf++;
        pc++;
    }
    printf(“%p\t%p\t%p\n\n”,pi,pf,pc);
    /* Muestra el estado final de los punteros */
    return 0;
}
//-----------------------------------------------------------------------
```

En este ejemplo observamos cómo cambia la dirección a la que apuntan los diferentes punteros mediante el uso del incremento según el tipo asociado a cada puntero.

Ejemplo

```
//-----------------------------------------------------------------------
#include<stdio.h>
//-----------------------------------------------------------------------
int main(void)
{
    long int l,*pl;
```

```
    char *pc;
    float f,*pf;
    pl=&l;
    scanf("%ld",&l);
    printf("\n%ld  %lx  %p\n",l,l,&l);
    /* Inicializa al puntero pc con la dirección del entero l, y
      recorre al entero mostrando su contenido byte a byte */
    for(pc=pl;pc<(char *)pl+4;pc++)
        printf("%p %2x\n",pc,*pc);
    pf=&f;
    scanf("%f",&f);
    printf("\n%f  %p\n",f,&f);
    /* Inicializa al puntero pc con la dirección del flotante f, y
      recorre al flotante mostrando su contenido byte a byte */
    for(pc=pf;pc<(char *)pf+4;pc++)
        printf("%p %2x\n",pc,*pc);
    return 0;
}
//-------------------------------------------------------------------
```

Podemos comprobar cómo se puede obtener el valor de una variable mediante el uso de diferentes tipos asociados a los punteros, consiguiendo su valor o su conformación byte a byte.

Otro dato que se puede restar a un puntero es el de otro puntero. En este caso el resultado será el de un entero, que significa la distancia en tipos asociados que separan a ambos punteros. Este valor puede ser positivo, indicando que el puntero que está como minuendo apunta a una dirección más alta de la memoria. Sii fuese negativo indicaría lo contrario.
Hay que tener mucho cuidado con esta interpretación ya que no expresa la distancia expresada en bytes.

Ejemplo

```
//-------------------------------------------------------------------
int main(void)
{
    int x,*pi,*pf,y,h;
    char *pc,*pd;
    float k,l,*pj,*pk;
    pi=&x;
    pf=&y;
    y=pf-pi;
    /* Obtiene la distancia entre los punteros pf y pi expresada en
      cantidades de enteros */
    pj=&l;
    pk=&k;
    x=pj-pk;
    /* Obtiene la distancia entre los punteros pj y pk expresada en
      cantidades de flotantes */
    pi=pj;
    pf=pk;
    y=pi-pf;
    /* Obtiene la distancia entre los punteros pj y pk expresada en
```

```
      cantidades de enteros, ya que los contenidos de estos
      punteros se le asignan a los punteros pi y pf que son
      punteros a enteros */
    pc=pj;
    pd=pk;
    h=pc-pd;
    /* Obtiene la distancia entre los punteros pj y pk expresada en
      cantidades de char, ya que los contenidos de estos punteros
      se le asignan a los punteros pc y pd que son punteros a
      char */
    return 0;
}
//--------------------------------------------------------------------
```

En este ejemplo podemos observar que la diferencia entre punteros nos indica la distancia entre las posiciones de memoria a la que apuntan, expresándola en cantidades de tipos asociados.

Ejemplo

```
//--------------------------------------------------------------------
#include<stdio.h>
//--------------------------------------------------------------------
int main(void)
{
    float *p,*q;
    int a;
    p=0x2000;
    q=0x200a;
    a=q-p;
    /* Calcula la distancia entre los punteros y los muestra en
      pantalla */
    printf("\n\n%d %p %p\n\n",a,p,q);
    return 0;
}
//--------------------------------------------------------------------
```

Vemos cómo se puede determinar la separación entre los punteros que apuntan a distintas posiciones de memoria.

Ejemplo

```
//--------------------------------------------------------------------
#include<stdio.h>
//--------------------------------------------------------------------
int main(void)
{
    int a[10],*pa1,*pa2,i,*pa3;
    /* Inicializa al vector */
    for (i=0;i<10;i++)
        a[i]=(i+1)*(i+1);
    /* Asigna al puntero pa1 la dirección de comienzo del vector y
```

```
    al puntero pa2 la dirección de la última posición del
    vector */
   pa1=a;
   pa2=&a[9];
   /* Obtiene la distancia entre los punteros */
   i=pa2-pa1;
   /* Asigna al puntero pa3 la dirección del segundo elemento del
    vector */
   pa3=pa1+i;
   return 0;
}
//------------------------------------------------------------------
```

Indica cómo se pueden obtener las posiciones extremas del vector y acceder a una posición intermedia del mismo.

La operación de suma de punteros no es posible realizarla ya que carece de significado.
Las operaciones de producto o cociente no son aceptadas por los punteros.
Otra imposibilidad es la de realizar operaciones utilizando datos de tipo flotante.
Las operaciones de desplazamiento o las operaciones lógicas no se pueden efectuar con los punteros.

Comparación

Los punteros admiten las operaciones de comparación (<,>,==,!=,<=,>=) que indican que el contenido de cada uno de los operandos es mayor, menor, igual o distinto del otro.
Si un puntero es mayor que el otro, esto indica que el mayor apunta a una dirección de memoria más alta que el otro. Si fueran iguales, ambos apuntarían a la misma dirección; y si fuesen distintos, ambos apuntarían a diferentes posiciones de memoria.
Hay que tener en cuenta que la comparación puede ser efectuada entre dos punteros, un puntero y un entero, o entre un puntero y la dirección de una variable.

Ejemplo

```
//------------------------------------------------------------------
#include<stdio.h>
//------------------------------------------------------------------
int main(void)
{
   int *pi,*pj;
   int i,j;
   pi=&i;
   pj=&j;
   /* Compara los punteros pi y pj determinando cuál de los dos es
    el mayor, es decir cual está apuntando a una posición de
    memoria más alta */
   if(pi>pj)
      printf("\npi es mayor a pj\n");
   else
                  printf("\npi es menor a pj\n");
   /* Compara al puntero pi con la dirección de la variable i para
    determinar si pi apunta o no a la variable i */
```

```
    if(pi==&i)
        printf("\npi es igual a &i\n");
    else
        printf("\npi no es igual a &i\n");
    /* Compara al puntero pi con la dirección 0x300 para saber si
      está apuntando a una dirección de memoria más alta o no */
    if(pi>0x300)
        printf("\npi apunta a una dirección mayor a 0x300\n");
    else
        printf("\npi apunta a una dirección menor a 0x300\n");
    return 0;
}
//------------------------------------------------------------------
```

Vemos cómo se pueden utilizar los operadores de comparación con punteros.

Punteros y vectores

Si se declara un vector, el compilador reservará la cantidad de memoria necesaria para el mismo, comenzando desde la posición cero (0). Cuando se hace referencia a un elemento determinado del vector, se utiliza el nombre seguido de un subíndice.
El compilador asigna al nombre del vector la dirección de comienzo de este, y el subíndice lo utiliza para desplazarse.
Si se utiliza un puntero para apuntar al comienzo del vector, se puede obtener el mismo resultado que utilizando el nombre del vector subindicado. Pero para realizar esto se debe utilizar la aritmética de punteros.
Como el nombre del vector contiene su dirección de comienzo, en realidad es un puntero constante, por lo que le podemos aplicar la aritmética de punteros y lograríamos así el mismo efecto.
En cambio, si se utiliza un subíndice con un puntero que apunte a la dirección de comienzo de un vector, obtendríamos el mismo resultado que subindicando un vector. Esto se denomina dualidad puntero-vector.

Ejemplo

```
//------------------------------------------------------------------
#include <stdio.h>
//------------------------------------------------------------------
int main(void)
{
    int V1[4],i;
    /* Intenta incrementar el nombre del vector generando un error,
      ya que el nombre del vector es un puntero constante a la
      dirección de comienzo del vector */
    for(i=0;i<4;i++)
        V1++;
    for(i=0;i<4;i++)
        scanf("%d",&V1[i]);
    for(i=0;i<4;i++)
        printf("%d\t",*(V1+i));
    return 0;
}
//------------------------------------------------------------------
```

En el ejemplo se muestra cómo se puede acceder a las diferentes posiciones del vector utilizando el nombre del mismo en forma subindicada o como puntero.

Ejemplo

```
//------------------------------------------------------------------
#include <stdio.h>
//------------------------------------------------------------------
int main(void)
{
    int V1[4],*pi,i;
    /* Inicializa al puntero con la dirección de comienzo
      del vector */
    pi=&V1[0];
    /* Recorre al vector utilizando el puntero pi para ingresar
      datos en este */
    for(i=0;i<4;i++)
        scanf("%d",pi+i);
    /* Recorre al vector mostrando su contenido utilizando al
      puntero en forma subindicada */
    for(i=0;i<4;i++)
        printf("%d\t",pi[i]);
    printf("\n\n");
    return 0;
}
//------------------------------------------------------------------
```

Aquí podemos ver cómo se accede al vector mediante un puntero, ya sea utilizándolo como puntero o subindicándolo.

Pasaje de parámetros por referencia en una función

Cuando se le envía un parámetro a una función se lo hace mediante una copia del valor que se envía. A este método se lo denomina pasaje por valor. En este caso, el parámetro que recibe la función es copiado en una variable que se declara en la misma y, como es una copia, cualquier modificación que se realice en esta variable no tendrá efecto sobre el valor de la enviada. Muchas veces no es esto lo que se quiere hacer. Por el contrario, cualquier modificación sobre el contenido de la copia se vea reflejado en la variable original. A este método se lo denomina pasaje por referencia.
El pasaje de parámetros por referencia en C se realiza utilizando punteros. Es decir que en el puntero se copia la dirección de la variable que se quiere modificar y en la función se utiliza el operador de indirección para acceder a su contenido. Esto hace que el valor que se modifique en la función aparecerá modificado en la variable original.
Es muy importante tener en cuenta que si se modifica el contenido del puntero, este apuntará a otra variable (una dirección de memoria distinta a la de la variable original) produciendo un resultado no esperado.

Ejemplo

```
//------------------------------------------------------------------
#include <stdio.h>
//------------------------------------------------------------------
```

```
void ingresar(int *V);
//-------------------------------------------------------------------
int main(void)
{
    int vec[10];
    /* Invoca a la función ingresar pasándole como parámetro la
      dirección de comienzo del vector vec */
    ingresar(vec);
    return 0;
}
//-------------------------------------------------------------------
void ingresar(int *V)
{
    /* Recibe como parámetro la dirección de comienzo del vector */
    int i;
    /* Ingresa valores al vector mediante el uso del puntero V */
    for(i=0;i<10;i++)
        scanf("%d",V+i);
}
//-------------------------------------------------------------------
```

En este ejemplo podemos ver cómo se accede al contenido de un vector utilizando una función. Para eso se le debe pasar como parámetro la dirección de comienzo del vector y para ello hay que enviarle a la función un puntero.

Punteros a string

Un puntero a string no es más que un puntero a char, debido a que un string no es nada más que un vector de char, pero que tiene la característica de que está finalizado por el carácter "null". Por lo que son aplicables los mismos conceptos que para cualquier otro vector.
Hay que tener en cuenta una particularidad de los punteros a string, esta es que si cunado se declara un puntero a char también se lo inicializa, se lo puede hacer de dos formas distintas:
La primera es asignarle la dirección de una variable, en este caso será como cualquier inicialización de un puntero.
La otra permite que se lo inicialice con un string, que será tomado como una constante.

En este caso el compilador reserva el espacio de memoria necesario para el string y lo ubica en esa posición, y le asigna al puntero la dirección de memoria correspondiente. Hay que tener cuidado en no modificar el contenido del puntero, ya que como el compilador reservo el espacio suficiente y no conocemos la dirección en donde esto ocurrió, al modificar al puntero perderemos la localización del string y no podrá ser vuelto a usar.
También hay que tener cuidado en no exceder el largo del string utilizado en la inicialización ya que solo está reservado el lugar necesario para este y si nos excedemos podemos ocupar un lugar destinado para otro valor y esto podría traer inconvenientes en la ejecución del programa.

Ejemplo

```
//-------------------------------------------------------------------
#include<stdio.h>
//-------------------------------------------------------------------
#define n 5
//-------------------------------------------------------------------
```

```
int main(void)
{
    /* Inicializamos a str con una cadena de caracteres */
    char str[]="int vec[n],*p,i;";
    /* Inicializa al puntero p con la dirección de comienzo del
      string */
    char *p=str;
    /* Se mueve al puntero dentro del string hasta encontrar la
      posición del carácter null */
    while(*p++);
    /* Muestra en pantalla el string y su largo mediante la
      diferencia entre el puntero y el nombre del string */
    printf("\nlongitud de \" %s \" es %d\n\n",str,p-str);
    return 0;
}
//-----------------------------------------------------------------
```

Podemos apreciar cómo podemos calcular la longitud del string utilizando un puntero.

Puntero a estructura

Una estructura es un tipo de dato definido por el usuario, y si después de definirlo se utiliza como tipo asociado de un puntero esto es totalmente valido.
La estructura como sabemos es un tipo de dato que contiene campos, distintos tipos válidos, y se puede acceder a ellos mediante el operador de acceso a miembro (.).
Esto es válido cuando se utilizan punteros ya que si un puntero apunta a una variable de tipo estructura a través del operador de indirección se puede acceder a su contenido, pero como el contenido de esta son campos, se le debería agregar el operador de acceso a miembro. Esto trae aparejado un inconveniente debido a la precedencia de los operadores, el operador de acceso a miembro tiene precedencia sobre el de indirección, por lo que habría que encerrar entre paréntesis al operador de indirección y al puntero para poder acceder a los campos de la estructura mediante el uso del operador de acceso a miembro.
Si bien este proceso no es dificultoso, trae aparejado inconvenientes en la escritura, ya que cuando se está generando el código, habría que tener en cuenta la precedencia de los operadores y esto puede producir errores.

Para solucionar este inconveniente se utiliza un nuevo operador, que se denomina operador apuntador de estructura (->).
Este operador permite el acceso directo a los campos mediante la utilización de un puntero.
Un cuidado que hay que tener es no confundir a una variable de tipo estructura con un vector, ya que el nombre de la variable, no contiene la dirección de esta y hay que utilizar el operador de direccionamiento para poder asignarle la dirección a un puntero.
Otro tipo de datos que también está definido por el usuario es la unión. Como este tipo de dato es muy parecido en su definición a la estructura pueden utilizarse las mismas herramientas para acceder a ella que se utilizan para las estructuras.

Ejemplo

```
//-----------------------------------------------------------------
#include <stdio.h>
//-----------------------------------------------------------------
struct A
```

```
{
    int a;
    char b;
    float c;
};
//-------------------------------------------------------------------
int main(void)
{
    struct A var,*ps;
    /* Accede al campo b de var mediante el operador punto */
    var.b='C';
    /* Asigna al puntero ps la dirección de la estructura */
    ps=&var;
    /* Accede al campo c de la estructura var mediante el puntero ps
      utilizando los operadores asterisco y punto */
    (*ps).c=3.14;
    /* Accede al campo a mediante el puntero ps utilizando el operador
      flecha */
    ps->a=7;
    return 0;
}
//-------------------------------------------------------------------
```

En este ejemplo podemos observar cómo se puede asignar un valor a un campo de una estructura utilizando un puntero

Puntero a puntero

Un puntero a puntero es solo un puntero que apunta a otro puntero. Esto es lo que se denomina indirección doble.

Para poder declararlo deberemos utilizar el operador de indirección dos veces y precedido por un tipo asociado.

tipo asociado ** nombre;
float **pf;
Podemos observar en el ejemplo que pf es un puntero a puntero a float, o lo que se podría interpretar como un puntero cuyo tipo asociado es float * (puntero a float).
El tamaño de un puntero a puntero es el mismo que el de un puntero, ya que va a almacenar una dirección de memoria.

Ejemplo

```
//-------------------------------------------------------------------
#include <stdio.h>
//-------------------------------------------------------------------
int main(void)
{
    int i,*pi,**ppi;
    i=65;
    /* Asigna al puntero pi la dirección de la variable i */
    pi=&i;
```

```
    /* Asigna al puntero a puntero ppi la dirección del
      puntero pi */
    ppi=&pi;
    /* Incrementa el contenido de la variable i en 4 mediante el
      uso del puntero pi */
    *pi+=4;
    /* Le asigna a la variable i el valor 1000 mediante el uso del
      puntero ppi */
    **ppi=1000;
    return 0;
}
//------------------------------------------------------------------
```

Se muestra cómo se puede acceder a una variable mediante una doble indirección con un puntero a puntero.

Vector de punteros

Un vector de punteros no es otra cosa que un vector cuyo tipo son punteros. Como todo vector su nombre es un puntero a la dirección de comienzo, pero en este caso como los elementos del vector son punteros, este es un puntero a puntero.

Ejemplo

```
//------------------------------------------------------------------
#include <stdio.h>
//------------------------------------------------------------------
int main(void)
{
    char c[5],*pc[5],*aux;
    int i,j;
    /* Asigna valores al vector c */
    for(i=0;i<5;i++)
        c[i]=getche();
    /* Inicializa al vector de punteros con las direcciones del
      vector c */
    for(i=0;i<5;i++)
        pc[i]=&c[i];
    /* Ordena al vector c utilizando el vector de punteros pc */
    for(i=0;i<4;i++)
        for(j=i+1;j<5;j++)
            if(*pc[i]>*pc[j])
            {
                aux=pc[i];
                pc[i]=pc[j];
                pc[j]=aux;
            }
    printf("\n");
    /* Muestra el contenido de lo apuntado por el vector de punteros
      (el vector c ordenado) y el contenido del vector c */
    for(i=0;i<5;i++)
```

```
        printf("%c  %c\n",*pc[i],c[i]);
    return 0;
}
//-----------------------------------------------------------------------
```

En este ejemplo se muestra cómo se puede utilizar un vector de punteros para poder ordenar un vector sin modificarlo mediante un vector de punteros.

TEMA 2

RECURSIVIDAD

La recursividad es una técnica que se puede emplear para solucionar ciertos tipos de algoritmos. Consiste en que la resolución del problema se basa en términos de sí mismo.
En general se puede decir que una función es recursiva si se invoca a sí misma.
Como hemos visto cuando se estudió el concepto de funciones, cada vez que se invoca a una función se guarda en el stack la dirección de retorno, y se generan las reservas de memoria de las variables locales de la función que se invoca. Cuando se utilizan funciones recursivas se almacenan en el stack la dirección de retorno y se crean las variables locales de esta, este proceso se repite tantas veces como la función sea invocada.

La recursividad tiene un inconveniente, que si no contiene una condición de salida se generara un algoritmo que no finaliza nunca, esto se puede detectar, ya que en algún momento dentro del funcionamiento del programa se desbordara el stack, produciendo un error en el funcionamiento del sistema operativo.

Tipos de recursividad

Podemos hablar de distintos tipos de recursividad, dependiendo de la forma en que ocurran las llamadas recursivas.

- Recursividad lineal
- Recursividad múltiple

Otra forma de clasificación es:

- Recursividad directa
- Recursividad indirecta

Recursividad lineal y recursividad múltiple

En la recursividad lineal, dentro de la función recursiva existe una única invocación a sí misma. En general, las funciones recursivas lineales son fácilmente transformadas a su versión iterativa.
Por el contrario, en la recursividad múltiple, la función se invoca a sí misma más de una vez dentro de una misma activación. Este tipo de funciones son más difíciles de llevar a su forma iterativa.

Recursividad directa e indirecta

Hasta ahora hemos visto sólo lo que llamamos recursividad directa: una función que se llama a sí misma.

Existe también la recursividad indirecta que es aquella que se produce cuando se tienen varias funciones que se llaman unas a otras formando un ciclo. Al igual que en la recursividad directa, es necesario contar con al menos un punto de parada de la recursividad.

Recursividad en comparación con iteración

Compararemos los dos enfoques y analizaremos porque el programador debe escoger un enfoque sobre el otro en una situación en particular. Tanto la iteración como la recursividad se basan en una estructura de control: la iteración utiliza una estructura de repetición, la recursividad utiliza una estructura de selección.
Tanto la iteración como la recursividad implican repetición: la iteración utiliza la estructura de repetición en forma explicita, la recursividad consigue la repetición mediante llamadas de función repetidas. La iteración y la recursividad ambas involucran una prueba de terminación: la iteración termina cuando falla la condición de continuación del ciclo, la recursividad termina cuando se reconoce un caso que puede resolverse sin recursividad.
La recursividad tiene muchas desventajas. La sobrecarga de llamadas de la función puede resultar costoso tanto en tiempo de procesador como en espacio de memoria. Cada llamada recursiva genera otra copia de las variables de la función, esto puede consumir gran cantidad de memoria. La iteración por lo regular ocurre dentro de una función por lo que no ocurre la sobrecarga de llamadas repetidas de función y asignación extra de memoria.

Ejemplo

```
//---------------------------------------------------------------------
#include<stdio.h>
//---------------------------------------------------------------------
int f(int);
//---------------------------------------------------------------------
int main(void)
{
    int i;
    /* Se invoca a la función recursiva f con un valor inicial de 3 */
    i=f(3);
    /* Se muestra en pantalla el valor retornado por f */
    printf("\ni=%d",i);
    return 0;
}
//---------------------------------------------------------------------
/* Definición de la función recursiva f */
int f(int i)
{
    /* Se muestra el valor que recibe la función f */
    printf("\ni=%d",i);
    /* Se invoca recursivamente a f con el valor de aumentado en uno */
    i=f(i+1);
    /* Retorna el valor final que toma la variable i */
    return i;
}
//---------------------------------------------------------------------
```

Podemos observar en este ejemplo que en la función f no existe ninguna condición de finalización. Esto provocaría el desborde del stack.

Ejemplo

```
//--------------------------------------------------------------------
#include <stdio.h>
//--------------------------------------------------------------------
int fun(int);
//--------------------------------------------------------------------
int main(void)
{
    int i;
    i=5;
    /* Se invoca a la función recursiva f con un valor inicial de 15 */
    i=fun(i*3);
    /* Se muestra en pantalla el valor retornado por f *|/
    printf("\n%d\n",i);
    return 0;
}
//--------------------------------------------------------------------
/* Definición de la función recursiva f */
int fun(int j)
{
    /* Se incrementa el valor recibido por la función en uno */
    j++;
    /* Se agrega una condición de finalización para la recursividad */
    /* Mientras el valor con que se invoca a fun sea menor a 20 se
      produce la recursividad en caso contrario se devuelve el valor
      de j */
    if(j<20)
        j=fun(j);
    return j;
}
//--------------------------------------------------------------------
```

En este ejemplo, cuando se invoca a la función fun con un valor mayor a 19, esta deja de invocarse a sí misma. En caso contrario, devuelve el valor con el que se la invocó incrementado en uno.

Ejemplo

```
//--------------------------------------------------------------------
#include <stdio.h>
//--------------------------------------------------------------------
long int fact(int);
//--------------------------------------------------------------------
int main(void)
{
    int x;
    long int f;
    /* Se ingresa el valor para el que se quiere calcilar el factorial */
    scanf("%d",&x);
    /* Se invoca a la función fact que permite calcular el factorial
      de un numero en forma recursiva */
```

```
    f=fact(x);
    /* Se muestra en pantalla el valor del factorial */
    printf("\n%d\t%ld",x,f);
    return 0;
}
//------------------------------------------------------------------------
/* Definición de la función recursiva fact que calcula el factorial
  de n */
long int fact(int n)
{
    /* Se inicializa a F con el valor de 1 que es 0! o 1! */
    long int F=1;
    /* Si el valor de n es mayor a 1 se invoca a fact con n
      decrementado en 1 */
    if(n>1)
        F=n*fact(n-1);
    /* Cuando el valor de n es menor o igual a 1 se retorna el valor
      de F que permite realizar el calculo del factorial */
    return F;
}
//------------------------------------------------------------------------
```

Esta función permite calcular el valor del factorial de un numero n. Este proceso se realiza teniendo en cuenta la definición del factorial.
Por ejemplo:

$$5!=5\cdot 4\cdot 3\cdot 2\cdot 1$$

Ejemplo

```
//------------------------------------------------------------------------
#include <stdio.h>
//------------------------------------------------------------------------
int fibo(int);
//------------------------------------------------------------------------
int main(void)
{
    int x,f;
    /* Se ingresa el valor para el que se quiere calcilar el término
      de la serie de Fibonacci */
    scanf("%d",&x);
    /* Se invoca a la función que obtiene el término de la seire */
    f=fibo(x);
    /* Se muestra el término de la serie */
    printf("\n%d\t%d",x,f);
    return 0;
}
//------------------------------------------------------------------------
/* Definición de la función recursiva fibo que calcula el término
    de la serie de Fibonacci */
int fibo(int n)
```

```
{
    int a;
    /* Condición de finalización de la recursividad */
    if(n<=2)
        a=1;
    else
        a=fibo(n-2)+fibo(n-1);
    return a;
}
//---------------------------------------------------------------------
```

Se puede observar cómo calcula el término enésimo de la serie en forma recursiva. Para realizar este cálculo se invoca a la función recursiva dos veces para cada término.

TEMA 3

UNIONES

Una unión permite compartir la misma posición de memoria para varias variables. Estas pueden ser del mismo tipo o de tipos diferentes.
Cuando son de distinto tipo, obviamente no tienen el mismo largo, y la de mayor tamaño es la que define el espacio de memoria que ocupa la unión.
A cada variable que compone la unión se la denomina “campo” (al igual que las estructuras).
Cada campo comienza en la misma posición de memoria y ocupa el largo correspondiente al tipo de la misma.
No se debe confundir una estructura con una unión. Si bien ambas son parecidas, hay una gran diferencia entre ellas. Las primeras ubican sus campos uno a continuación del otro, mientras que en las uniones los campos comparten la misma ubicación en la memoria de la computadora. Todos comienzan en la misma dirección de memoria.
Su importancia radica en que se pueden ingresar datos utilizando un tipo de variable y extraerlos utilizando otro. Otra de sus ventajas es que cuando uno posee una cantidad limitada de memoria para datos se puede utilizar la misma ubicación de memoria para guardar datos de distinto tipo, con el cuidado de no superponer diferentes variables al mismo tiempo.

Declaración

La declaración de una unión se puede realizar de tres diferentes formas.

```
union etiqueta { tipo campo_1;
                 tipo campo_2;
                 tipo campo_3;
                 . . .
                 . . .
                 tipo campo_n;} lista_de_variables;

union { tipo campo_1;
        tipo campo_2;
        tipo campo_3;
        . . .
        . . .
        tipo campo_n;} lista_de_variables;

union etiqueta { tipo campo_1;
                 tipo campo_2;
                 tipo campo_3;
                 . . .
                 . . .
                 tipo campo_n;};
```

La primera es la más general. Se debe utilizar la palabra reservada “unión”, luego un nombre del tipo “etiqueta”, a continuación las llaves de apertura y de cierre ({ }), que se utilizan para encerrar los campos, y las variables que tendrán el tipo de la unión definido, en la lista_de_variables. Esta forma se la utiliza para declarar uniones en forma global o local pera cada función.
En la segunda forma se omite la etiqueta, pero aquí debe incluirse la lista_de_variables. En el caso que no se especifique la lista no se podrá utilizar, debido a que no es posible realizar una identificación de ésta.
La última consiste en omitir la lista_de_variables, al igual que en las estructuras, ésta es la más utilizada pudiéndosela utilizar en cualquier función o también definirla de forma global, para ello solo basta con declarar el tipo en el momento en que se precise. Para declarar una o más variables se debe realizar lo siguiente:

union etiqueta lista_de_variables;

Ejemplo

```
#include<stdio.h>
union data{  short int valor;
                  char dato[2];}      ent[100];
void ingreso(void);
void main(void)
    {
        union data salida;
        ------------
        ------------
        ------------
    }
void ingreso(void)
    {
union data entrada;
        ------------
        ------------
        ------------
        }
```

Podemos observar que en el ejemplo se declaró una unión denominada data, que contiene dos campos: uno de tipo “short int” y el otro es un vector de dos posiciones de tipo “char”.
Se declara un vector de 100 posiciones llamado “ent” como variable global, que es visto por las funciones “main” e “ingreso”.
En cada una de las funciones se declara una variable del tipo “union data” que solo es vista por la función en la que fue declarada.

```
#include <stdio.h>
union {  int A;
          float B;} calc;
void soluc(void);
void main(void)
    {
        union {     int A;
                    float B;} calc;
        ------------
        ------------
        ------------
```

```
    }
void soluc(void)
    {
        union {     int A;
                    float B;} clave;
        ------------
        ------------
        ------------
    }
```

Aquí podemos observar que las uniones definidas en este ejemplo no tienen la etiqueta y por lo tanto no se puede reutilizar este tipo con otras variables. Si se quiere usar un tipo de unión igual al definido en forma global hay que redefinirlo. Pero realizando esto los tipos de las uniones no son idénticos y no es posible asignar una variable a otra.
También se puede observar que se han definido en cada función uniones iguales en forma local a ellas.

```
#include <stdio.h>
union un1     {     char A;
                    unsigned long B;};
void disposit(void);
void main(void)
   {
        union un1 clase;
        ------------
        ------------
        ------------
   }
void disposit(void)
   {
        union un1 sensado;
        ------------
        ------------
        ------------
    }
```

Aquí tenemos una definición de un tipo global de unión denominado “un1”. Es global y por lo tanto este tipo es visto por todas las funciones del programa.
Como tiene una etiqueta que identifica al tipo de la unión, se pueden declarar variables de tipo local o global solamente utilizando la siguiente sintaxis: union un1 seguido por el o los nombres de las variables que se deseen declarar bajo ese tipo.

Referencia y asignación a campos en una unión

Para hacer referencia a un campo de una unión, se utiliza el operador punto o miembro, invocando primero el nombre de la unión, seguido del operador punto y por último el nombre del campo, de forma similar a la que se utiliza en una estructura.
Cuando se desea realizar una asignación a un campo en la unión se utiliza el signo igual (=), al ejecutar esto último solo debe tenerse en cuenta que las variables a asignar deben ser necesariamente del mismo tipo.
Se puede asignar una unión a otra, pero ambas deben ser del mismo tipo, ambas deben tener la misma declaración o etiqueta.

Ejemplo

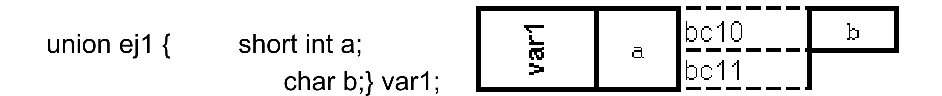
```
#include<stdio.h>
#include<conio.h>
/* Se define una unión */
union alfa      {     short int A[2];
                          char B[4];
                          float C;};
void main(void)
   {
         /* Se declara una variable del tipo “union alfa” */
         union alfa dato;
         /* Asigno e imprimo valores al campo “A” */
         dato.A[0]=6529;
         dato.A[1]=30287;
         clrscr();
         printf(“\n\ndato.A[0]=%d\tdato.A[1]=%d”,dato.A[0],dato.A[1]);
         printf(“\nEn formato hexadecimal”);
         printf(“\ndato.A[0]=%X\tdato.A[1]=%X”,dato.A[0],dato.A[1]);
         /* Asigno e imprimo valores al campo “B” */
         dato.B[0]=’A’;
         dato.B[1]=’B’;
         dato.B[2]=’C’;
         dato.B[3]=’D’;
         printf(“\n\ndato.B[0]=%c\tdato.B[1]=%c”,dato.B[0],dato.B[1]);
         printf(“\tdato.B[2]=%c\tdato.B[3]=%c”.dato.B[2],dato.B[3]);
         printf(“\nEn formato ASCII”);
         printf(“\ndato.B[0]=%d\tdato.B[1]=%d\tdato.B[2]=%d\t\
                  dato.B[3]=%d”,dato.B[0],dato.B[1],dato.B[2],dato.B[3]);
         printf(“\nEn formato hexadecimal”);
         printf(“\ndato.B[0]=%X\tdato.B[1]=%X\tdato.B[2]=%X\t\
                  dato.B[3]=%X”,dato.B[0],dato.B[1],dato.B[2],dato.B[3]);
         /* Asigno e imprimo un valor al campo “C” */
         dato.C=3.123e-10;
         printf(“\n\ndato.C=%G”,dato.C);
         printf(“\n\nPresione una tecla para continuar”);
         getch();
         /* Asigno e imprimo valores al campo “A” */
         dato.A[0]=6529;
         dato.A[1]=30287;
         clrscr();
         printf(“\n\ndato.A[0]=%d\tdato.A[1]=%d”,dato.A[0],dato.A[1]);
         printf(“\nEn formato hexadecimal”);
         printf(“\ndato.A[0]=%X\tdato.A[1]=%X”,dato.A[0],dato.A[1]);
         /* visualizo los valores en el campo “B” */
         printf(“\n\nSin cambiar los valores examinamos el campo “”B”””);
         printf(“\n\ndato.B[0]=%c\tdato.B[1]=%c”,dato.B[0],dato.B[1]);
         printf(“\tdato.B[2]=%c\tdato.B[3]=%c”,dato.B[2],dato.B[3]);
         printf(“\nEn formato ASCII”);
         printf(“\ndato.B[0]=%d\tdato.B[1]=%d\tdato.B[2]=%d\t\
                  dato.B[3]=%d”,dato.B[0],dato.B[1],dato.B[2],dato.B[3]);
         printf(“\nEn formato hexadecimal”);
         printf(“\ndato.B[0]=%X\tdato.B[1]=%X\tdato.B[2]=%X\t\
```

```
                dato.B[3]=%X”,dato.B[0],dato.B[1],dato.B[2],dato.B[3]);
    /* visualizo los valores en el campo “C” */
    printf(“\n\nSin cambiar los valores examinamos el campo “”C”””);
    printf(“\n\ndato.C=%G”,dato.C);
    printf(“\n\nPresione una tecla para continuar”);
    getch();
    clrscr();
    /* Asigno e imprimo valores al campo “B” */
    dato.B[0]=’A’;
    dato.B[1]=’B’;
    dato.B[2]=’C’;
    dato.B[3]=’D’;
    printf(“\n\ndato.B[0]=%c\tdato.B[1]=%c”,dato.B[0],dato.B[1]);
    printf(“\tdato.B[2]=%c\tdato.B[3]=%c”,dato.B[2],dato.B[3]);
    printf(“\nEn formato ASCII”);
    printf(“\ndato.B[0]=%d\tdato.B[1]=%d\tdato.B[2]=%d\t\
                dato.B[3]=%d”,dato.B[0],dato.B[1],dato.B[2],dato.B[3]);
    printf(“\nEn formato hexadecimal”);
    printf(“\ndato.B[0]=%X\tdato.B[1]=%X\tdato.B[2]=%X\t\
                dato.B[3]=%X”,dato.B[0],dato.B[1],dato.B[2],dato.B[3]);
    /* visualizo los valores en el campo “A” */
    printf(“\n\nSin cambiar los valores examinamos el campo “”A”””);
    printf(“\n\ndato.A[0]=%d\tdato.A[1]=%d”,dato.A[0],dato.A[1]);
    printf(“\nEn formato hexadecimal”);
    printf(“\ndato.A[0]=%X\tdato.A[1]=%X”,dato.A[0],dato.A[1]);
    /* visualizo los valores en el campo “C” */
    printf(“\n\nSin cambiar los valores examinamos el campo “”C”””);
    printf(“\n\ndato.C=%G”,dato.C);
    printf(“\n\nPresione una tecla para continuar”);
    getch();
    clrscr();
    /* Asigno e imprimo un valor al campo “C” */
    dato.C=3.123e-10;
    printf(“\n\ndato.C=%G”,dato.C);
    /* visualizo los valores en el campo “A” */
    printf(“\n\nSin cambiar los valores examinamos el campo “”A”””);
    printf(“\n\ndato.A[0]=%d\tdato.A[1]=%d”,dato.A[0],dato.A[1]);
    printf(“\nEn formato hexadecimal”);
    printf(“\ndato.A[0]=%X\tdato.A[1]=%X”,dato.A[0],dato.A[1]);
    /* visualizo los valores en el campo “B” */
    printf(“\n\nSin cambiar los valores examinamos el campo “”B”””);
    printf(“\n\ndato.B[0]=%c\tdato.B[1]=%c”,dato.B[0],dato.B[1]);
    printf(“\tdato.B[2]=%c\tdato.B[3]=%c”,dato.B[2],dato.B[3]);
    printf(“\nEn formato ASCII”);
    printf(“\ndato.B[0]=%d\tdato.B[1]=%d\tdato.B[2]=%d\t\
                dato.B[3]=%d”,dato.B[0],dato.B[1],dato.B[2],dato.B[3]);
    printf(“\nEn formato hexadecimal”);
    printf(“\ndato.B[0]=%X\tdato.B[1]=%X\tdato.B[2]=%X\t\
                dato.B[3]=%X”,dato.B[0],dato.B[1],dato.B[2],dato.B[3]);
    printf(“\n\nPresione una tecla para terminar”);
    getch();
}
```

En este ejemplo se muestra como se puede asignar y referirse a un campo de la unión. Se asignan valores por separado a cada campo y luego se los imprime.
Seguidamente se asignan valores a un campo y luego se imprimen todos los campos. Este procedimiento permite ver como se comparten las posiciones de memoria en una unión, pudiendo visualizar que si se utiliza erróneamente un campo, se puede perder la información contenida en otro.

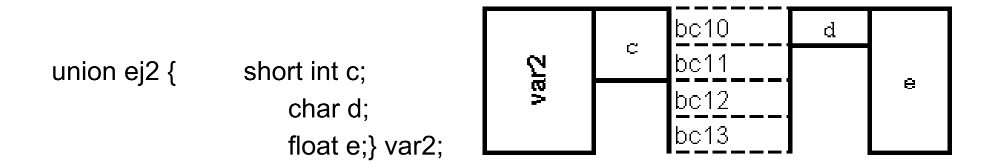
Ubicación y tamaño de los campos de una unión en la memoria

Una unión se ubica en la zona de memoria perteneciente al segmento de datos. A partir de la dirección donde comienza la unión se ubican los campos. Si bien todos comienzan en ese preciso lugar, el largo de la unión, lo determina el campo de mayor tamaño.
Por ejemplo, en la siguiente unión:

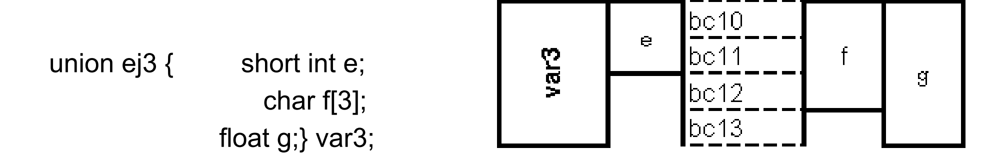


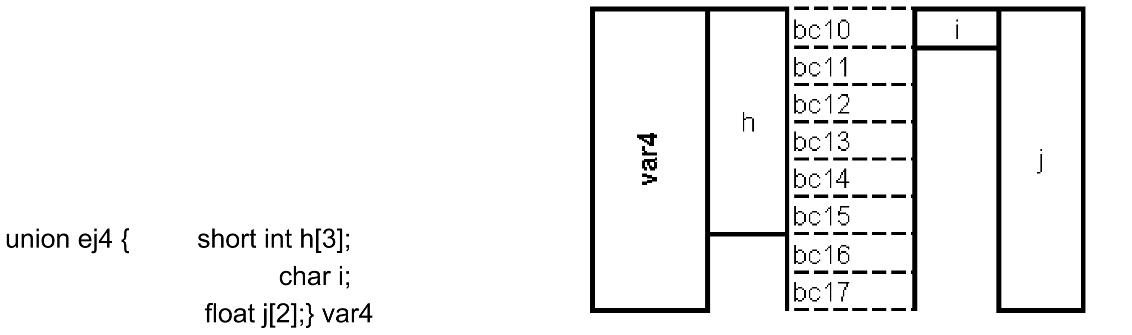
Podemos observar que esta compuesta por dos campos uno de tipo short int y el otro de tipo char. Ambos se ubicaran a partir de la misma dirección de memoria, manteniendo su dimensión típica, es decir, el campo short int ocupara dos bytes mientras que el char ocupa solamente uno, dando un tamaño total para la unión de 2 bytes.

Si analizamos la unión que sigue a continuación:



Encontraremos que los tres campos también comienzan en la misma dirección pero como en este caso tenemos además un float cambiará solamente el largo de la misma, siendo en este caso de 4 bytes, debido a que el tipo de mayor tamaño es el float.
Cuando nos encontramos en presencia de un vector, este comenzará en la dirección de comienzo de la unión, ocupando los lugares que corresponden a su largo. Esto no significa que determina el largo de la unión: lo determinará el campo de mayor tamaño. Veamos los siguientes ejemplos:





En el primero nos encontramos con un vector de char con un largo de 3 posiciones, una variable entera y una de punto flotante. Como la de mayor tamaño de los tres campos es el float éste determinará el largo total de la estructura.
En el segundo nos encontramos con dos vectores uno short int de 3 posiciones (6 bytes) y otro de tipo float de 2 posiciones (8 bytes), y además una variable de tipo char (1 byte). Aquí encontraremos que el vector de punto flotante es el de mayor largo, dando como resultado que este tamaño será el de la unión.

Pasaje y retorno de miembros de una unión a una función

Los campos de una unión son en sí mismos variables como cualquier otra, tienen un tipo asociado y un identificador válido (nombre). Esto hace que se pueda retornar un campo de una unión desde una función o se pueda enviar el mismo como parámetro de la misma.

Ejemplo

```
#include<stdio.h>
#include<conio.h>
/* Se define el tipo de la union */
union alfa    {    short int x;
                        char y[10];
                        float z;};
/* Se declaran los prototipos de las funciones */
short int suma(short int , short int );
short int ingreso(void);
void main(void)
   {
        /* Se declaran las variables de tipo union */
        union alfa A,B;
        clrscr();
        /* Se invoca a la función “ingreso” */
        A.x=ingreso();
        B.x=ingreso();
        printf(“\n%d+%d=”,A.x,B.x);
        /* Se invoca a la función “suma” */
        A.x=suma(A.x,B.x);
        printf(“%d\n”,A.x);
        /* Se limpia el buffer del teclado */
        fflush(stdin);
        gets(A.y);
```

```
        if(A.y[2]>'F')
                {
                    A.z=3.14;
                    scanf("%f",&B.z);
                    printf("\n\nPerimetro = %f",A.z*B.z);
                }
   }
/* Se define a la función "ingreso"     *
 * ésta devuelve un valor de tipo " short int" *
short int ingreso(void)
   {
        short int a;
        scanf("%d",&a);
        return(a);
   }
/* Se define a la función "suma"         *
 * ésta devuelve un valor de tipo " short int"  *
 * y recibe dos parámetros de tipo " short int" *
short int suma(short int a, short int b)
   {
        short int c;
        c=a+b;
        return(c);
   }
```

En este ejemplo se define una unión que contiene tres campos: uno de tipo " short int" denominado "x", otro de tipo "char" llamado "y" que es un vector de 10 posiciones y un campo "float", cuyo nombre es "z".
Primero se asignan al campo "x" de las variables "A" y "B" los valores que retorna la función "ingreso", esta permite la entrada de un valor " short int" utilizando la función "scanf".
Luego se envían a la función "suma" el campo "x" de las uniones "A" y "B" y el resultado de ésta se le asigna al campo "x" de la variable "A", el valor devuelto por esta función es el resultado de sumar ambos parámetros.

Pasaje y retorno de uniones a funciones

Se puede realizar la transferencia de una unión como parámetro de una función procediendo de forma similar a lo que se realiza con una estructura, indicando el tipo de la unión en la declaración y prototipo de la función. Además, la declaración de la misma debe realizarse en forma global, dado que si así no se realizase no sería vista por las funciones a las cuales se les transfiere su valor.
De igual forma, se puede retornar una unión desde una función, indicando el tipo de la misma como parámetro de retorno de ella. En la sentencia "return", dentro de la función, se debe retornar una variable del tipo de la unión utilizada para retornar.
Hay que tener cuidado de que solo se pueden pasar y retornar uniones del mismo tipo hacia y desde una función.

Ejemplo

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
```

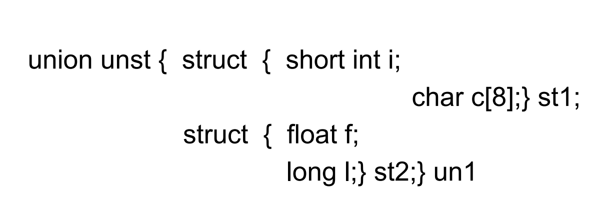
```
/* Se define una unión con dos campos uno "int" y el otro "char" */
union alfa    {    int x;
                        char y;};
/* Se declaran los prototipos de las funciones */
union alfa ingreso(char);
void imprime(union alfa,char);
void main(void)
    {
        /* Se declaran las variables */
        union alfa A;
        char c;
        do
            {
                /* Menú de opciones */
                clrscr();
                printf("\n\n\t\t\tIngrese una opción ");
                printf("\n\n (C) ingreso de un carácter");
                printf("\n\n (E) ingreso de un entero ");
                c=getche();
                printf("\n\n");
                /* Se invoca a la función "ingreso" */
                A=ingreso(c);
                /* Se invoca a la función "imprime" */
                imprime(A,c);
                printf("\n\n\nDesea ingresar otro valor (S/N) ");
                c=getche();
            }
        while((c=='s')||(c=='S'));
    }
/* Se define a la función "ingreso"  *
 * devuelve una unión de tipo "alfa" *
 * recibe un "char"                  */
union alfa ingreso(char b)
    {
        union alfa C;
        printf("Ingrese un ");
        if((b=='c')||(b=='C'))
                {
                    printf(" carácter ");
                    C.y=getche();
                }
            else
```

```
            {
                printf(" entero ");
                scanf("%d",&C.x);
            }
        printf("\n\n");
        /* Devuelve el valor de la unión */
        return(C);
    }
/* Se define la función "imprime"          *
 * recibe una unión de tipo alfa llamada "B" *
 * y un char denominado "c", devuelve "void" */
void imprime(union alfa B,char c)
    {
        if((c=='c')||(c=='C'))
                printf("\n\nUsted ingreso el carácter %c",B.y);
            else
                printf("\n\nUsted ingreso el entero %d",B.x);
    }
```

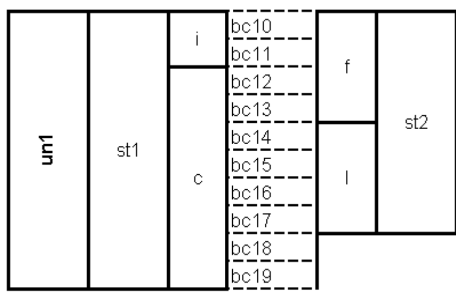
En este ejemplo se puede ver cómo se recibe una unión como valor retornado por una función y cómo una función recibe como parámetro el valor de una unión.
En la función “ingreso” se observa cómo es el procedimiento de retorno de una unión. Esta es invocada en “main” y el valor retornado se le asigna a una unión denominada “A”.
En “main’ también se llama a la función “imprime” pasándole los parámetros de los cuales uno es del tipo de la unión “alfa”, mediante el envío de la variable “A”.

Uniones de estructuras

Una unión puede contener como campos cualquier tipo de dato válido, por lo que dentro de ella también pueden existir una o varias estructuras. Cuando existen estructuras dentro de una unión es lo que se llama “unión de estructuras”.
En este caso los campos de la unión se encuentrán empaquetados dentro de cada estructura, ahora el tamaño de la unión está dado por la estructura mayor, aquella que ocupa mayor cantidad de bytes en la memoria.
Ahora las estructuras comienzan en la misma dirección, solapándose una a la otra. Veamos un ejemplo de este tipo de uniones.



```
union unst {  struct  {  short int i;
                                char c[8];} st1;
              struct  {  float f;
                         long l;} st2;} un1
```



Podemos observar que no se solapan las variables de cada estructura entre sí, pero si lo hacen ambas estructuras.
Debemos recordar que para acceder a un campo de la unión se debe utilizar el operador punto. Utilizando uno solo accederíamos a una estructura. Para acceder a una variable deberíamos utilizar dos operadores punto, el primero para acceder a la estructura y el segundo para poder referenciar el campo de la estructura.
Si deseamos referirnos al campo “l” de la estructura “st2” que pertenece a la unión “un1” lo debemos hacer de la siguiente forma:

```
un1.st2.l=948;
```

Ejemplo

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
/* Largo máximo del vector y largo de la pantalla */
#define     MAX  100
#define     PANT 6
/* Definición de las estructuras que integraran a la unión */
struct tiem      {   unsigned char min,seg;};
struct libros  {   char ident;
                        unsigned int num;
                        char titu[50],autor[50],edit[20];
                        unsigned int pag;};
struct discos{   char ident;
                        unsigned int num;
                        char inte[30],gene[20];
                        unsigned char tem;
                        struct tiem dur;};
/* Definición de la unión de estructuras */
union datos      {   struct libros lib;
                        struct discos dis;};
/* Declaración de los prototipos de las funciones */
void bolib(union datos indice[]);
void inlib(union datos indice[]);
void lilib(union datos indice[]);
void indis(union datos indice[]);
void bodis(union datos indice[]);
void lidis(union datos indice[]);
/* Declaración de una variable global e inicialización de la misma */
int cant=0;
void main(void)
   {
```

```
        /* Declaración de las variables de "main" */
        char op;
        union datos indice[MAX];
        do
            {
                /* Menú de opciones */
                clrscr();
                printf("\n\n\t\t\tMenú principal");
                printf("\n\n\t (I) Ingresar libros");
                printf("\n\t (N) Ingresar discos");
                printf("\n\t (B) Borrar libros");
                printf("\n\t (O) Borrar discos");
                printf("\n\t (L) Listar libros");
                printf("\n\t (T) Listar discos");
                printf("\n\t (Q) Salir");
                printf("\n\n\n\t\t\tIngrese una opción ");
                op=getche();
                switch(op)
                    {
                        case 'B':  case 'b':  bolib(indice);
                                                      break;
                        case 'I':  case 'i':  inlib(indice);
                                                      break;
                        case 'L':  case 'l':  lilib(indice);
                                                      break;
                        case 'N':  case 'n':  indis(indice);
                                                      break;
                        case 'O':  case 'o':  bodis(indice);
                                                      break;
                        case 'T':  case 't':  lidis(indice);
                                                      break;
                    }
            }
        while((op!='Q')&&(op!='q'));
    }
/* definición de la función "bolib" que se encargara de eliminar  *
 * los datos de uno o mas libros                                   */
void bolib(union datos indice[])
    {
        char c,titu[50];
        int i,j;
        do
```

```
            {
                clrscr();
                printf("\t\tIngrese el titulo del libro ");
                fflush(stdin);
                gets(titu);
                for(i=0;(i<=cant)&&(strcmp(indice[i].lib.titu,titu));i++);
                if(i>cant)
                        printf("\n\n\t\tEl libro no fue encontrado ");
                    else
                        {
                            printf("\n\t\tAutor : %s",indice[i].lib.autor);
                            printf("\n\t\tEditorial : %s",indice[i].lib.edit);
                            printf("\n\nEstá seguro (S/N)? ");
                            c=getche();
                            if((c=='s')||(c=='S'))
                                    {
                                        for(j=i;j<cant;j++)
                                            indice[j]=indice[j+1];
                                        cant--;
                                    }
                        }
                printf("\n\nDesea borrar otro libro (S/N)? ");
                c=getche();
            }
        while((c=='s')||(c=='S'));
    }
/* Definición de la función "inlib" que se encargara de ingresar *
 * uno o mas libros                                             */
void inlib(union datos indice[])
    {
        struct libros L;
        int i;
        char c;
        do
            {
                if(cant<MAX)
                        {
                            L.ident='L';
                            for(i=cant;(indice[i].lib.ident!='L')&&i;i--);
                            if(i)
                                    L.num=indice[i].lib.num+1;
```

```
                    else
                        L.num=1;
                do
                    {
                        clrscr();
                        fflush(stdin);
                        printf("\n\tIngrese el titulo del libro ");
                        gets(L.titu);
                        printf("\n\tIngrese el autor del libro ");
                        gets(L.autor);
                        printf("\n\tIngrese la editorial del libro ");
                        gets(L.edit);
                        printf("\n\tIngrese la cantidad de páginas ");
                        printf("del libro ");
                        scanf("%u",&L.pag);
                        printf("\n\nLos datos son correctos (S/N) ? ");
                        c=getche();
                    }
                while((c!='s')&&(c!='S'));
                cant++;
                indice[cant].lib=L;
            }
        else
            printf("\n\nNo puede ingresar un nuevo libro");
      printf("\n\nDesea ingresar otro libro (S/N)? ");
      c=getche();
    }
  while((c=='s')||(c=='S')&&(cant<MAX));
}
/* Definición de la función "lilib" que se encargara de listar *
 * los libros                                                   */
void lilib(union datos indice[])
{
    int i,j;
    for(i=0;i<cant;i+=PANT)
      {
        clrscr();
        for(j=0;(j<PANT)&&((i+j)<=cant);j++)
          {
            if(indice[i+j].lib.ident=='L')
                {
```

```
                        printf("\n\nNúmero : %d ",indice[i+j].lib.num);
                        printf("Titulo : %s ",indice[i+j].lib.titu);
                        printf("\nAutor : %s ",indice[i+j].lib.autor);
                        printf("Editorial : %s ",indice[i+j].lib.edit);
                        printf("Cantidad de páginas : %u ",\
                        indice[i+j].lib.pag);
                    }
                }
            printf("\n\nPresione una tecla para continuar");
            getch();
        }
    }
/* Definición de la función "indis" que será la encargada de   *
 * ingresar uno o mas discos                                   */
void indis(union datos indice[])
    {
        struct discos L;
        int i;
        char c;
        do
            {
            if(cant<MAX)
                {
                    L.ident='D';
                    for(i=cant;(indice[i].dis.ident!='D')&&i;i--);
                    if(i)
                        L.num=indice[i].dis.num+1;
                    else
                        L.num=1;
                    do
                        {
                        clrscr();
                        fflush(stdin);
                        printf("\n\tIngrese el interprete del disco ");
                        gets(L.inte);
                        printf("\n\tIngrese el genero del disco ");
                        gets(L.gene);
                        printf("\n\tIngrese la cantidad de temas ");
                        printf("del disco ");
                        scanf("%u",&L.tem);
                        printf("\n\tIngrese la duración del ");
                        printf("disco (mm:ss) ");
```

```
                        scanf("%u:%u",&L.dur.min,&L.dur.seg);
                        printf("\n\nLos datos son correctos (S/N) ? ");
                        c=getche();
                    }
                while((c!='s')&&(c!='S'));
                cant++;
                indice[cant].dis=L;
            }
        else
            printf("\n\nNo puede ingresar un nuevo disco");
        printf("\n\nDesea ingresar otro disco (S/N)? ");
        c=getche();
    }
    while((c=='s')||(c=='S')&&(cant<MAX));
}
/* Definición de la función "bodis" que será la encargada de borrar *
 * uno o mas discos                                                 */
void bodis(union datos indice[])
{
    char c,titu[50];
    int i,j;
    do
    {
        clrscr();
        printf("\t\tIngrese el interprete del disco ");
        fflush(stdin);
        gets(titu);
        for(i=0;(i<=cant)&&(strcmp(indice[i].lib.titu,titu));i++);
        if(i>cant)
            printf("\n\n\t\tEl disco no fue encontrado ");
        else
            {
                printf("\n\t\tGenero : %s",indice[i].dis.gene);
                printf("\n\nEsta seguro (S/N)? ");
                c=getche();
                if((c=='s')||(c=='S'))
                    {
                        for(j=i;j<cant;j++)
                            indice[j]=indice[j+1];
                        cant--;
                    }
            }
```

```
                printf("\n\nDesea borrar otro disco (S/N)? ");
                c=getche();
            }
        while((c=='s')||(c=='S'));
    }
/* Definición de la función "lidis" que será la *
 * encargada de listar en pantalla a los discos */
void lidis(union datos indice[])
    {
        int i,j;
        for(i=0;i<cant;i+=PANT)
            {
                clrscr();
                for(j=0; (j<PANT)&&((i+j)<=cant);j++)
                    {
                        if(indice[i+j].dis.ident=='D')
                                {
                                printf("\n\nNúmero : %d ",indice[i+j].dis.num);
                                printf("Interprete : %s ",indice[i+j].dis.inte);
                                printf("\nGenero : %s ",indice[i+j].dis.gene);
                                printf("Temas : %u ",indice[i+j].dis.tem);
                                printf("Duración : %u:%u ",\
                                            indice[i+j].dis.dur.min,\
                                            indice[i+j].dis.dur.seg);
                            }
                }
            printf("\n\nPresione una tecla para continuar");
            getch();
        }
}
```

Se puede observar la utilización de una unión de estructuras para guardar los datos de dos objetos diferentes, como ser libros y discos. Estos tienen características particulares que no permiten poder guardarlos en un único vector.
Se utilizan 6 funciones para poder realizar las diferentes operaciones elegidas en el menú de opciones que se despliega en la función “main”. En éstas se permite el ingreso de datos, eliminar datos y el listado de los mismos.

Estructuras de uniones

En este caso debemos hacer referencia a una estructura en la cual algunos o todos los pueden ser uniones. Utilizando este tipo de datos se puede ver que los campos que pertenecen a la estructura son uniones y se ubican una a continuación de la otra. Las variables que se solapan son las que están dentro de cada unión.

```
struct stun  {   union {     short int i1;
                             char c;}un1;
                 union {     short int i2[2];
                             long l;}un2;} st1;
```

st1	un1	i1	bc10	c
			bc11	
	un2	i2	bc12	l
			bc13	
			bc14	
			bc15	

Encontramos en el ejemplo anterior que las variables “c” e “i1” y “l” e “i2” están ocupando las mismas posiciones de memoria, pero la unión “un1” esta ubicada a continuación de “un2”. Para referenciar un campo de la estructura se debe utilizar el operador punto, al igual que en la unión de estructuras se usará un operador para invocar a una unión y dos operadores punto para invocar a la variable de la unión.

```
st1.un2.i2[1]=65;
```

En el ejemplo encontramos que se asigna el valor 65 al campo “i2[1]” de la unión “un2” de la estructura “st1”.

Ejemplo

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
/* Largo máximo del vector y largo de la pantalla */
struct libros  {    char ident;
                    unsigned int num;
                    char titu[50],autor[50],edit[20];
                    unsigned int pag;};
struct discos{      char ident;
                    unsigned int num;
                    char inte[30],gene[20];
                    unsigned char tem;
                    struct tiem dur;};
/* Definición de la unión de estructuras */
union datos     {   struct libros lib;
                    struct discos dis;};
/* Declaración de los prototipos de las funciones */
void bolib(union datos indice[]);
void inlib(union datos indice[]);
void lilib(union datos indice[]);
void indis(union datos indice[]);
void bodis(union datos indice[]);
void lidis(union datos indice[]);
/* Declaración de una variable global e inicialización de la misma */
int cant=0;
void main(void)
```

```
    {
        /* Declaración de las variables de "main" */
        char op;
        union datos indice[MAX];
        do
            {
                /* Menú de opciones */
                clrscr();
                printf("\n\n\t\t\tMenú principal");
                printf("\n\n\t (I) Ingresar libros");
                printf("\n\t (N) Ingresar discos");
                printf("\n\t (B) Borrar libros");
                printf("\n\t (O) Borrar discos");
                printf("\n\t (L) Listar libros");
                printf("\n\t (T) Listar discos");
                printf("\n\t (Q) Salir");
                printf("\n\n\n\t\t\tIngrese una opción ");
                op=getche();
                switch(op)
                    {
                        case 'B':  case 'b':   bolib(indice);
                                                        break;
                        case 'I':   case 'i':   inlib(indice);
                                                        break;
                        case 'L':  case 'l':   lilib(indice);
                                                        break;
                        case 'N':  case 'n':   indis(indice);
                                                        break;
                        case 'O':  case 'o':   bodis(indice);
                                                        break;
                        case 'T':  case 't':   lidis(indice);
                                                        break;
                    }
            }
        while((op!='Q')&&(op!='q'));
    }
/* definición de la función "bolib" que se encargara de eliminar  *
 * los datos de uno o mas libros                                  */
void bolib(union datos indice[])
    {
        char c,titu[50];
        int i,j;
```

```
        do
           {
              clrscr();
              printf("\t\tIngrese el titulo del libro ");
              fflush(stdin);
              gets(titu);
              for(i=0;(i<=cant)&&(strcmp(indice[i].lib.titu,titu));i++);
              if(i>cant)
                    printf("\n\n\t\tEl libro no fue encontrado ");
                 else
                    {
                       printf("\n\t\tAutor : %s",indice[i].lib.autor);
                       printf("\n\t\tEditorial : %s",indice[i].lib.edit);
                       printf("\n\nEstá seguro (S/N)? ");
                       c=getche();
                       if((c=='s')||(c=='S'))
                             {
                                for(j=i;j<cant;j++)
                                   indice[j]=indice[j+1];
                                cant--;
                             }
                    }
              printf("\n\nDesea borrar otro libro (S/N)? ");
              c=getche();
           }
        while((c=='s')||(c=='S'));
     }
/* Definición de la función "inlib" que se encargara de ingresar *
 * uno o mas libros                                             */
void inlib(union datos indice[])
     {
        struct libros L;
        int i;
        char c;
        do
           {
              if(cant<MAX)
                    {
                       L.ident='L';
                       for(i=cant;(indice[i].lib.ident!='L')&&i;i--);
                       if(i)
                             L.num=indice[i].lib.num+1;
```

```
                        else
                            L.num=1;
                    do
                        {
                            clrscr();
                            fflush(stdin);
                            printf("\n\tIngrese el titulo del libro ");
                            gets(L.titu);
                            printf("\n\tIngrese el autor del libro ");
                            gets(L.autor);
                            printf("\n\tIngrese la editorial del libro ");
                            gets(L.edit);
                            printf("\n\tIngrese la cantidad de páginas ");
                            printf("del libro ");
                            scanf("%u",&L.pag);
                            printf("\n\nLos datos son correctos (S/N) ? ");
                            c=getche();
                        }
                    while((c!='s')&&(c!='S'));
                    cant++;
                    indice[cant].lib=L;
                }
            else
                printf("\n\nNo puede ingresar un nuevo libro");
        printf("\n\nDesea ingresar otro libro (S/N)? ");
        c=getche();
    }
    while((c=='s')||(c=='S')&&(cant<MAX));
  }
/* Definición de la función "lilib" que se encargara de listar *
 * los libros                                              */
void lilib(union datos indice[])
  {
    int i,j;
    for(i=0;i<cant;i+=PANT)
      {
        clrscr();
        for(j=0;(j<PANT)&&((i+j)<=cant);j++)
          {
            if(indice[i+j].lib.ident=='L')
                {
                    printf("\n\nNúmero : %d ",indice[i+j].lib.num);
```

```
                        printf("Titulo : %s ",indice[i+j].lib.titu);
                        printf("\nAutor : %s ",indice[i+j].lib.autor);
                        printf("Editorial : %s ",indice[i+j].lib.edit);
                        printf("Cantidad de páginas : %u ",\
                        indice[i+j].lib.pag);
                    }
                }
            printf("\n\nPresione una tecla para continuar");
            getch();
        }
    }
/* Definición de la función "indis" que será la encargada de   *
 * ingresar uno o mas discos                                   */
void indis(union datos indice[])
    {
        struct discos L;
        int i;
        char c;
        do
            {
            clrscr();
            printf("\t\tIngrese el interprete del disco ");
            fflush(stdin);
            gets(titu);
            for(i=0;(i<=cant)&&(strcmp(indice[i].lib.titu,titu));i++);
            if(i>cant)
                printf("\n\n\t\tEl disco no fue encontrado ");
              else
                {
                    printf("\n\t\tGenero : %s",indice[i].dis.gene);
                    printf("\n\nEsta seguro (S/N)? ");
                    c=getche();
                    if((c=='s')||(c=='S'))
                        {
                            for(j=i;j<cant;j++)
                                indice[j]=indice[j+1];
                            cant--;
                        }
                }
            printf("\n\nDesea borrar otro disco (S/N)? ");
            c=getche();
            }
```

```
            while((c=='s')||(c=='S'));
        }
    /* Definición de la función "lidis" que será la *
     * encargada de listar en pantalla a los discos */
    void lidis(union datos indice[])
        {
            int i,j;
            for(i=0;i<cant;i+=PANT)
                {
                    clrscr();
                    for(j=0; (j<PANT)&&((i+j)<=cant);j++)
                        {
                            if(indice[i+j].dis.ident=='D')
                                    {
                                        printf("\n\nNúmero : %d ",indice[i+j].dis.num);
                                        printf("Interprete : %s ",indice[i+j].dis.inte);
                                        printf("\nGenero : %s ",indice[i+j].dis.gene);
                                        printf("Temas : %u ",indice[i+j].dis.tem);
                                        printf("Duración : %u:%u ",\
                                                    indice[i+j].dis.dur.min,\
                                            indice[i+j].dis.dur.seg);
                                }
                    }
                printf("\n\nPresione una tecla para continuar");
                getch();
            }
}
```

Este ejemplo permite realizar el ingreso y la visualización de los datos de vehículos para poder realizar las ventas. Utiliza dos funciones para ello: una llamada “ingreso” y la otra denominada “listado”. Ambas reciben el vector y la cantidad de datos ingresados hasta el momento.
El vector en donde se ingresan los datos es un vector de estructuras en el cual dos de los campos utilizados son uniones. Éstas se utilizan para guardar los datos según la característica elegida, tipo de automóvil o forma de la venta.
Las uniones se denominan “clase” y “fina”. La unión “clase” se compone de dos campos “dat” y “dat1” que a su vez son estructuras, llamadas “coupe” y “sedan”. Ellas guardaran los datos correspondientes a las características particulares de cada tipo.
La unión “fina” contiene dos campos, uno de tipo “float” y el otro, una estructura que permite guardar los datos correspondientes a la cantidad de cuotas y el valor de cada cuota.
Se utilizan variables auxiliares para permitir el ingreso de valores en formato numérico a variables de tipo “char” y para poder realizar el ingreso de datos de tipo “float”.

TEMA 4

CAMPOS DE BITS

Un campo de bits es una estructura particular que permite asignar valores binarios a cada campo de la estructura.
Cada campo de la estructura es una variable binaria del largo que se fije según se desee.

Declaración

La declaración de un campo de bits se realiza de una de tres diferentes formas.

```
struct etiqueta {   tipo entero campo_1: cantidad de bits;
                    tipo entero campo_2: cantidad de bits;
                    tipo entero campo_3: cantidad de bits;
                    . . .
                    . . .
                    tipo entero campo_n: cantidad de bits;} lista_de_variables;

struct {   tipo entero campo_1: cantidad de bits;
           tipo entero campo_2: cantidad de bits;
           tipo entero campo_3: cantidad de bits;
           . . .
           . . .
           tipo entero campo_n: cantidad de bits;} lista_de_variables;

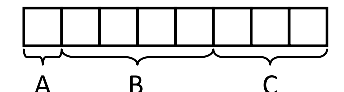
struct etiqueta {   tipo entero campo_1: cantidad de bits;
                    tipo entero campo_2: cantidad de bits;
                    tipo entero campo_3: cantidad de bits;
                    . . .
                    . . .
                    tipo entero campo_n: cantidad de bits;};
```

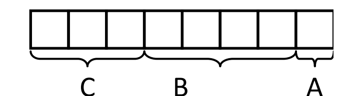
Podemos observar que la diferencia entre la declaración de una estructura y un campo de bits consiste en que solamente se puede utilizar un tipo de dato entero para declarar al campo y que se especifica el largo en bits de dicho campo.
Hay que tener en cuenta que la máxima cantidad de bits que se pueden asignar solo depende del tamaño de la variable entera que se utiliza en su definición.
En todo lo demás se comporta como una estructura.
Con referencia al tipo, se pueden utilizar el “char”, “int” o “long” pero no son convenientes debido a que el valor que se observará cuando el bit más significativo tome el valor uno, será un valor distinto al deseado. Esto es debido a que cuando ocurre tal caso en una variable entera, este bit corresponde al signo de dicha variable. Para solucionar este aspecto es conveniente utilizar el modificador “unsigned”.

El espacio asignado a cada uno de los campos de bits depende del tamaño del tipo asociado al campo de bits, de tal forma que ubica cada uno de los campos en forma sucesiva hasta llegar a ocupar todos los bits del largo del tipo asociado comenzado desde el bit menos significativo al más significativo. Hay que tener en cuenta que este proceso no es portable, entonces deberemos determinar la forma en que se ubican dichos campos en cada sistema en particular. En el caso este proceso no fuese de esta forma lo será en forma inversa pudiendo corregirlo durante el proceso.
Supongamos el siguiente campo de bits:

```
Struct datos {   unsigned char A:3;
                 unsigned char B:4;
                 unsigned char C:1;};
```

El tamaño total de la estructura sería de un byte, debido a que el total de los bits individuales de cada uno de los campos es igual al largo de un char (1 byte). Estos se ubicarían de estas posibles formas:





Si la cantidad de bits definidos es menor al del tipo asociado, estos quedarán vacíos. Si ocurriese que una variable no cupiese en el lugar disponible, se agregará un nuevo tamaño del tipo asociado.

Ejemplo

```
#include <stdio.h>
struct dato {  unsigned int A:5;
                    unsigned int B:9;} valores;
float valmed(void);
void main(void)
    {
        struct dato ingreso;
        ------------
        ------------
        ------------
    }

float valmed(void);
    {
        struct dato aux;
        ------------
        ------------
        ------------
    }
```

Aquí podemos encontrar que la estructura dato se define como un campo de bits, en ésta se encuentra que se define un campo de un ancho de 5 bits denominado “A” y otro de un ancho de 9 bits llamado “B”.
También encontramos que se define una variable global de este tipo, “valores” y dos variables locales una a “main” y la otra a “valmed”.

```
#include <stdio.h>
struct {  unsigned char A:3;
            unsigned char B:1;
            unsigned char C:2;}   calc;
void soluc(void);
void main(void)
   {
struct {  unsigned char A:3;
            unsigned char B:1;
            unsigned char C:2;}   datos[25];
       ------------
       ------------
       ------------
   }
void soluc(void)
   {
struct {  unsigned char A:3;
            unsigned char B:1;
            unsigned char C:2;}clave;
       ------------
       ------------
       ------------
   }
```

En este se definen tres campos de bits idénticos entre sí, pero como no se lo asignó una etiqueta a cada uno de ellos, los campos de bits no son del mismo tipo.
El campo “A” tiene una longitud de 3 bits, el “B” tan solo de un bit y el “C” de dos bits.

```
#include <stdio.h>
struct cpo1 { unsigned long A:13;
                  unsigned long B:15;
                  unsigned long C:7;};
void disposit(void);
void main(void)
   {
      struct cpo1 clase;
      ------------
      ------------
      ------------
        }
void disposit(void)
   {
      struct cpo1 sensado;
      ------------
      ------------
      ------------
   }
```

Podemos ver aquí que se define un tipo de dato de forma global que contiene tres campos de bits, uno de 13, otro de 15 y el último de 7.
Todas las variables fueron declaradas locales a las funciones del tipo “struct cpo1”.

Referencia y asignación de valores en un campo de bits

Para poder referenciar un valor en un campo de bits es necesario, primero, escribir el nombre de la estructura que contiene el campo de bits a utilizar; luego, escribir el operador punto; y por último, invocar el nombre del campo de bits que se quiera utilizar. Este procedimiento es el mismo que se utiliza para llamar a un campo en una estructura.
Cuando se quiere asignar un valor a un campo de bits se debe utilizar el operador de asignación (=) escribiendo en el lado derecho del operador el valor numérico deseado o la variable que se le quiere asignar.
Hay que tener la precaución de utilizar una variable para asignársela a un campo de bits. Únicamente se puede realizar esta operación cuando se asignan campos de igual tamaño.

Ejemplo

```
#include<stdio.h>
#include<conio.h>
/* Se definen los campos de bits */
struct alfa     {   unsigned char A:1;
                        unsigned char B:2;
                        unsigned char C:3;};
struct beta     {   unsigned char A:3;
                        unsigned char B:2;
                        unsigned char C:1;};
struct gamma  {   unsigned char A:5;
                        unsigned char B:2;
                        unsigned char C:3;
                        unsigned char D:4;};
union delta     { struct alfa D;
                        struct beta E;
                        struct gamma F;
                        char G[2];};
void main(void)
   {
        char a;
        /* Se declara un campo de bits y una unión para visualizar el *
        * comportamiento de los campos de bits                         */
        struct alfa b;
        union delta e;
        clrscr();
        /* Se examinan los valores posibles de las variables binarias  *
        * verificando que en caso de exederse el máximo valor posible *
        * vuelve al primer valor válido                               */
        for(b.A=0,a=0;a<4;b.A++,a++)
            printf("%d   ",b.A);
        printf("\n");
        for(b.B=0,a=0;a<8;b.B++,a++)
            printf("%d   ",b.B);
        printf("\n");
        for(b.C=0,a=0;a<16;b.C++,a++)
            printf("%d   ",b.C);
        printf("\n\nPresione una tecla para continuar ");
        getch();
```

```
    clrscr();
    /* Limpia el contenido del vector de la unión y lo imprime */
    e.G[0]=0;
    e.G[1]=0;
    printf("\ne.g[0]=%d   e.G[1]=%d",e.G[0],e.G[1]);
    /* Asigna valores a los campos de bits y los imprime */
    e.D.A=1;
    e.D.B=2;
    e.D.C=5;
    printf("\ne.D.A=%d  e.D.A=%d  e.D.C=%d",e.D.A,e.D.B,e.D.C);
    printf("\ne.G[0]=%X  e.G[1]=%X",e.G[0],e.G[1]);
    e.E.C=1;
    e.E.B=2;
    e.E.A=5;
    printf("\ne.E.A=%d  e.E.A=%d  e.E.C=%d",e.E.A,e.E.B,e.E.C);
    printf("\ne.G[0]=%X  e.G[1]=%X",e.G[0],e.G[1]);
    e.F.A=18;
    e.F.B=2;
    e.F.C=5;
    e.F.D=11;
    printf("\ne.F.A=%X  e.F.A=%X  ",e.F.A,e.F.B);
    printf("e.F.C=%X  e.F.D=%X",e.F.C,e.F.D);
    printf("\ne.G[0]=%X  e.G[1]=%X",e.G[0],e.G[1]);
    printf("\n\nPresione una tecla para continuar ");
    getch();
}
```

Vemos en este ejemplo cómo se puede asignar un valor y referenciarse a un campo de bits. También se puede observar que las variables del tipo de campo de bits pueden tomar solo los valores permitidos según su tamaño. Cuando la variable supera su valor máximo automáticamente vuelve a comenzar con el valor mínimo.
Se puede verificar cómo se ubican los campos de bits en la memoria mediante el uso de una unión, revisando los valores que se le ingresan a las variables binarias e imprimiendo los bytes correspondientes.

TEMA 5

OPERADORES A NIVEL DE BITS

Los operadores a nivel de bits son aquellos que trabajan modificando los valores de los bits individuales, es decir, que actúan bit a bit sobre el valor binario guardado en la posición de memoria sobre la que trabajan.
Los operadores convencionales se diferencian de éstos porque actúan sobre la variable completa, evaluando su contenido sin importar los estados de los bits que la componen.

Operador and (&)

El operador and (&) realiza la función del producto lógico entre dos elementos pero analizado bit a bit. Se diferencia del operador relacional and (&&) que también realiza la misma operación, pero este ultimo lo ejecuta evaluando el contenido completo de la variable y no lo realiza analizando cada par de bits en particular.
Recordemos que el resultado de una función “and” es el siguiente:

a	b	a&b
0	0	0
0	1	0
1	0	0
1	1	1

Este operador tiene la ventaja de poder poner al descubierto el o los bits deseados en una variable utilizando la máscara adecuada (valor numérico elegido per el usuario para poder realizar la tarea deseada) y poniendo en cero los bits que carecen de interés.

Operador or (|)

Este realiza la función lógica or (suma lógica), evaluando los operandos bit a bit. Se diferencia del operador relacional **or (||)** dado que éste analiza a los operandos según su contenido total.
Reveamos los valores que toma la función “or”

<table>
<tr><th>a</th><th>b</th><th>a | b</th></tr>
<tr><td>0</td><td>0</td><td>0</td></tr>
<tr><td>0</td><td>1</td><td>1</td></tr>
<tr><td>1</td><td>0</td><td>1</td></tr>
<tr><td>1</td><td>1</td><td>1</td></tr>
</table>

Posee también la capacidad de poder enmascarar bits en forma individual, poniendo en uno los bits no deseados y dejando con el valor que tenían aquellos que son de interés.

Operador xor (^)

Este operador realiza la función lógica o exclusiva.
También permite enmascarar bits al igual que los anteriores.
Su resultado es el siguiente:

a	b	a^b
0	0	1
0	1	0
1	0	0
1	1	1

Operador not (~)

Realiza la negación de los bits correspondientes: estos invierten su valor. Cuando un bit tiene el valor uno, al negarlo se transforma en cero.
El efecto de esta función es el siguiente:

a	~ a
0	1
1	0

Operador desplazamiento izquierdo (<<)

Este operador realiza el desplazamiento de los bits de una variable a la izquierda perdiendo el contenido del bit más significativo y agregando ceros a los bits ubicados a la derecha. La cantidad de bits a desplazar depende del valor indicado a la derecha del símbolo <<. Esta operación debe utilizarse siempre a la izquierda del operador de asignación (=).

Operador desplazamiento derecho (>>)

Realiza el desplazamiento de la cantidad de bits a la derecha, perdiendo el bit menos significativo e insertando cero a los bits de la izquierda.
En esta operación se debe utilizar siempre con el símbolo de asignación (=).

Ejemplo

```
#include<stdio.h>
#include<conio.h>
void main (void)
   {
        unsigned int nro,aux;
        clrscr();
        do
```

```
        {
            clrscr();
            printf("\n\n\t\tIngrese un número entero : ");
            scanf("%d",&nro);
            printf("\n\n\t\tEl valor en binario es : ");
            /* Imprime un número en formato binario  *
            * para el cálculo utiliza una máscara y *
            * utiliza para ello la operación and    *
            * y el desplazamiento de bits          */
            for(aux=0x8000;aux>0;aux>>=1)
              if(nro&aux)
                   printf("1");
                 else
                   printf("0");
             printf("\n\n\nDesea continuar (S/N)? ");
        }
              while(getche()!='N');
  }
```

En este ejemplo se obtiene la representación de un número en formato binario. Para ello se realiza una operación "and' entre el número dado y una máscara que tiene solo un bit en uno y el resto con valor cero.
Se procede a comprobar el valor del resultado de dicha operación e imprimir un uno en caso de que la comparación sea verdadera y un cero, en caso contrario. Se va desplazando el uno de la máscara hacia la derecha en un lugar de forma tal de poder escribir el número binario en el orden correcto, desde el bit más significativo hasta el menos significativo.

```
#include<stdio.h>
#include<conio.h>
void main (void)
  {
      int i,j;
      clrscr();
      do
        {
            clrscr();
            printf("\n\n\t\tIngrese un número entero : ");
            scanf("%d",&i);
            /* Imprime el número ingresado en formato hexadecimal */
            printf("\n\t\tEl número en hexadecimal es %04X",i);
            /* Calcula el complemento a dos del número ingresado */
            j=~i+1;
            /* Imprime el número en complemento a dos en el formato *
             * decimal y en el formato hexadecimal                 */
            printf("\n\n\t\tEl número en complemento a dos es %d",j);
            printf("\n\n\t\tEn formato hexadecimal es %04X",j);
            printf("\n\n\nDesea continuar (S/N)? ");
        }
      while(getche()!='N');
  }
```

En este se muestra cómo convertir un número en su correspondiente valor en complemento a dos. Para ello se utiliza el operador “not” y luego se le suma uno al resultado de la operación de negación del número correspondiente.
Esta notación se utiliza en lenguaje “C” para expresar un número negativo. La representación del número ingresado y de su resultado en complemento a dos se expresan en formato decimal y en formato hexadecimal. Se puede observar que los resultados en formato decimal son iguales en módulo pero de signo contrario; en cambio, en formato hexadecimal se encuentra que los resultados son distintos entre sí, pero hay que ver que ambos difieren en el bit más significativo.

```
#include<stdio.h>
#include<conio.h>
void conv(unsigned char,char,unsigned char,unsigned char);
void conv1(char,unsigned char,unsigned char);
void conv2(unsigned char,char,unsigned char,unsigned char);
void pasbin(unsigned char);
void main(void)
    {
        unsigned char a,i,b,c;
        do
            {
                clrscr();
                printf("\n\n\t\tIngrese un número entero : ");
                scanf("%d",&a);
                printf("\n\t\tIngrese otro número entero : ");
                scanf("%d",&b);
                /* Calcula todas las operaciones a nivel de bit entre *
                 * los dos números ingresados                        */
                printf("\n\n\t\t%d | %d = %d",a,b,a|b);
                conv(a,'|',b,a|b);
                printf("\n\n\t\t%d & %d = %d",a,b,a&b);
                conv(a,'&',b,a&b);
                printf("\n\n\t\t%d ^ %d = %d",a,b,a^b);
                conv(a,'^',b,a^b);
                printf("\n\n\t\t~%d = %d",a,~a);
                conv1('~',a,~a);
                printf("\n\n\t\t~%d = %d",b,~b);
                conv1('~',b,~b);
                printf("\n\n\t\t\tPresione una tecla para continuar");
                getch();
                clrscr();
                printf("\n\n");
                /* Calcula los desplazamientos a derecha *
                 * de los valores ingresados            */
```

```
                for(i=0;i<9;i++)
                    {
                        c=a>>i;
                        printf("\n\t\t%d >> %d = %d ",a,i,c);
                        conv2(a,'>',i,c);
                    }
                printf("\n\n");
                for(i=0;i<9;i++)
                    {
                        c=b>>i;
                        printf("\n\t\t%d >> %d = %d ",b,i,c);
                        conv2(b,'>',i,c);
                    }
                printf("\n\n\t\t\tPresione una tecla para continuar");
                getch();
                clrscr();
                printf("\n\n");
                /* Calcula los desplazamientos a izquierda *
                 * de los valores ingresados              */
                for(i=0;i<9;i++)
                    {
                        c=a<<i;
                        printf("\n\t\t%d << %d = %d ",a,i,c);
                        conv2(a,'<',i,c);
                    }
                printf("\n\n");
                for(i=0;i<9;i++)
                    {
                        c=b<<i;
                        printf("\n\t\t%d << %d = %d ",b,i,c);
                        conv2(b,'<',i,c);
                    }
                printf("\n\nDesea continuar (S/N)? ");
            }
        while(getche()!='N');
    }
/* Escribe un número en formato binario */
void pasbin(unsigned char nro)
    {
        unsigned char aux;
        for(aux=0x80;aux>0;aux>>=1)
                    if(nro&aux)
```

```
                    printf("1");
                else
                    printf("0");
    }
/* Rutina de impresión de las operaciones en formato binario */
void conv(unsigned char a,char b,unsigned char c,unsigned char d)
    {
        printf("\t\t");
        pasbin(a);
        printf(" %c ",b);
        pasbin(c);
        printf(" = ");
        pasbin(d);
    }
/* Rutina de impresión de las operaciones en formato binario */
void conv1(char a,unsigned char b,unsigned char c)
    {
        printf("\t\t%c",a);
        pasbin(b);
        printf(" = ");
        pasbin(c);
    }
/* Rutina de impresión de las operaciones en formato binario */
void conv2(unsigned char a,char b,unsigned char c,unsigned char d)
    {
        printf("\t\t");
        pasbin(a);
        printf(" %c%c %d = ",b,b,c);
        pasbin(d);
    }
```

En este podemos ver los resultados de aplicar las operaciones a nivel de bit entre dos números que se ingresan desde el teclado. Estos valores se expresan tanto en formato decimal como en forma binaria. Para expresar los valores en forma binaria se utiliza la función “pasbin”, que es igual a la que se utilizó en el primer ejemplo.
Las funciones “conv”, “conv1” y “conv2” son funciones que realizan la impresión formateada de los datos en binario con la expresión de la misma.

TEMA 6

ARCHIVOS

Un archivo es un conjunto de datos que se intercambian en un dispositivo de entrada o salida. Este conjunto de datos puede ser de cualquier tipo; incluso entre ellos no tiene por qué haber un orden o tipo predeterminado.
Como los dispositivos de entrada y salida pueden ser diversos, es decir, disco, memoria de almacenamiento, teclado, pantalla, impresora, etc., se necesita poder normalizar el acceso a los distintos tipos de dispositivos, de forma tal que no sea necesario conocer el hardware especifico al que se quiere acceder. Para esto, el lenguaje C utiliza un elemento de almacenamiento intermedio denominado “stream”.
Este elemento se ubica entre la computadora y el dispositivo de almacenamiento, de forma tal que el procesador y el dispositivo siempre vean un bloque de memoria, haciendo que el sistema operativo sea el encargado del manejo del hardware determinado.
Un stream es una estructura en la memoria de la computadora que se va a encargar del manejo del sistema de archivos. Tiene un buffer que permite almacenar los datos que provienen o se dirigen al archivo. Este buffer transfiere los datos en un sentido u otro al archivo, pero teniendo en cuenta que cuando se transfieren los datos de la computadora al dispositivo, primero se llena el buffer y luego se transfiere todo el contenido al dispositivo de salida. Cuando se transfieren los datos del archivo a la computadora se llena el buffer y luego la computadora los va tomando en la medida en que se lo necesite.
Este proceso tiene el inconveniente de que si ocurre un inconveniente en el sistema, se pierden los datos almacenados en el buffer.
Debemos tener en cuenta que para poder trabajar con archivos se debe crear el stream, y cuando se finaliza el uso del archivo, se lo debe destruir. Estos procesos se realizan mediante la apertura y cierre del archivo.
El lenguaje C genera, cuando se inicia, un programa y se abren tres stream, que son los denominados stream estándar. Estos son: stdin, stdout y stderr. El stream stdin es el que está relacionado con la entrada estándar del programa (teclado); el stdout es el que se utiliza para la salida estándar (monitor), y el stderr es el que se utiliza para el manejo de los mensajes de error, esto se realiza utilizando como dispositivo al monitor.
Vamos a utilizar funciones cuyos prototipos están ubicados en el header stdio.h. También se va a utilizar la estructura FILE que está definida en este header.

Apertura de un archivo

Para poder abrir un archivo se utiliza la función “fopen” cuyo prototipo se encuentra en el header “stdio.h”.

Su prototipo es:

FILE *fopen(char *nombre,char *modo)

Esta función recibe dos parámetros punteros a char, es decir que recibe dos strings.

El primero es el nombre del archivo con el que se quiere trabajar. Hay que tener en cuenta que lo que se denomina nombre en un archivo es un término que consta de cuatro partes: unidad, ruta, nombre y extensión.
Cuando se refiere a la unidad, es la unidad de disco en la que se encuentra el archivo en cuestión, a la unidad se la puede omitir dentro de este parámetro pero en este caso hay que tener en cuenta que va a utilizar la unidad actual, es decir, la unidad en la que en ese momento está parado el sistema operativo.
El término ruta se refiere a la ubicación del archivo dentro del árbol de directorios de la unidad especificada. Este término también se puede omitir, pero el archivo se ubicará en la ruta actual del sistema, la ruta en la que está posicionado en ese momento el sistema operativo.
El nombre no se puede omitir, ya que es lo que identifica al archivo. En cuanto a la extensión, este término puede ser omitido; pero si se lo omite, será un archivo que no tendrá extensión.
El segundo parámetro que recibe la función es el modo que se refiere a la forma en que se va a abrir el archivo y también al tipo de archivo.
Los archivos se pueden abrir en tres formas diferentes: lectura (r), escritura (w) y para añadir (a).
Cuando se abre en el modo de lectura, se entiende que el archivo debe existir con anterioridad, para que pueda ser leído, si el archivo no existiese no se lo podría leer y entonces se produciría un error.
En el modo de lectura no se crea un archivo si este no existiese.
El modo de escritura se utiliza para escribir un nuevo archivo, es decir que se crea un archivo.
Si se quisiese abrir un archivo existente en modo escritura, el contenido de este sería destruido, guardándose solo lo que se escribió luego de la apertura.
Si se quisiera abrir un archivo para poder agregarle nuevos datos, habría que hacerlo para añadir.
Tanto en el modo de escritura como en el de lectura, los archivos se abren al comienzo. Si se quiere abrir para añadir, se abren al final del mismo.
Existe un modo de apertura que se denomina extendida, que se indica mediante el carácter +.
Cuando se utiliza este modo, los archivos pueden ser leídos como escritos, sin importar en el modo en que se abrieron; pero hay que tener en cuenta que el modo de apertura es el que va a dominar la forma de abrirlo. Es decir si, un archivo existente se abre en el modo w+, el contenido original seria destruido porque se lo abrió en modo escritura, aunque se lo puede también leer.
En cuanto al tipo de archivo, existen dos: el archivo de texto o el binario.
Un archivo de texto es aquel en el que su contenido son caracteres, es decir todos los valores que se almacenan podrán ser leídos e interpretados fácilmente mediante un editor de texto.
En cambio, cuando se utiliza un archivo binario, se guarda el contenido de la memoria en el archivo sin hacer ningún tipo de corrección en los datos. Esto hace que no sea fácil poder interpretar el contenido del archivo mediante el uso de un editor de texto. También si el tamaño de los datos que se almacenan es el mismo que se utiliza en la memoria. En cambio, en los archivos de texto los datos numéricos pueden tener distinta longitud en el archivo, ya que se convierten los datos que se almacenan en la memoria por los caracteres numéricos que los representan. De esta forma el tamaño dependerá del valor numérico que se quiere almacenar.

ATRIBUTO	SIGNIFICADO
r ó rt	Abre un archivo de texto para lectura. Si no existe producirá un error.
w ó wt	Crea un archivo de texto para escritura. En el caso de que exista destruye el contenido.
a ó at	Abre un archivo de texto para añadir. Si no existe crea uno, si existe agrega los datos al final.
Rb	Abre un archivo binario para lectura. Si no existe producirá un error.
Wb	Crea un archivo binario para escritura. En el caso de que exista destruye el contenido.

ATRIBUTO	SIGNIFICADO
Ab	Abre un archivo binario para añadir. Si no existe crea uno, si existe agrega los datos al final.
r+ ó r+t	Abre un archivo de texto para lectura/escritura. Si no existe no lo crea.
w+ ó w+t	Crea un archivo de texto para lectura/escritura. Si existe destruye los contenidos del archivo existente.
a+ ó a+t	Abre o crea un archivo de texto para lectura/escritura.
r+b	Abre un archivo binario para lectura/escritura. Si no existe no lo crea.
w+b	Crea un archivo binario para lectura/escritura. Si existe destruye los contenidos del archivo existente.
a+b	Abre o crea un archivo binario para lectura/escritura.

Cuando se quiere abrir un archivo puede suceder que no se lo pueda abrir por diferentes motivos, todos ellos denominados errores o que ocurra lo que uno desea, que lo abra correctamente. En el primer caso la función fopen devolverá un puntero NULL; en cambio, cuando se abre correctamente, devuelve la dirección de comienzo de la estructura FILE.
La estructura FILE está definida en el header stdio.h.
Cada vez que se abre un archivo se debe verificar que el archivo se abrió correctamente, para poder tomar la decisión correspondiente.

Ejemplo:

```
//-----------------------------------------------------------------
#include <stdio.h>
#include <stdlib.h>
//-----------------------------------------------------------------
int main(void)
{
    /* Declara un puntero a FILE para poder manejar el archivo */
    FILE *fp;
    /* Invoca a la función fopen para abrir el archivo de texto
       nuevoarc.txt en modo de lectura */
    fp=fopen(“nuevoarc.txt”,”r”);
    /* Si el archivo no existe no lo puede abrir y para eso se
       chequea el valor que devuelve fopen */
    if(fp==NULL)
    {
        /* Si no se pudo abrir se tomara la acción adecuada */
        printf(“\n\nError de apertura\n\n”);
        exit(1);
    }
    /* Si se abrió correctamente se procede a cerrar el archivo */
    fclose(fp);
    return 0;
}
//-----------------------------------------------------------------
```

En este ejemplo se puede observar cómo se abre un archivo, si no se puede abrir el archivo, fopen devolverá un NULL, en este caso se deberá tomar una acción adecuada ya que el archivo no se pudo abrir.

Ejemplo:

```
//--------------------------------------------------------------------
#include <stdio.h>
#include <stdlib.h>
//--------------------------------------------------------------------
int main(void)
{
    FILE *fp;
    /* Invoca a la función fopen para abrir el archivo de texto
       nuevoarc.txt en modo de escritura */
    fp=fopen(“nuevoarc.txt”,”w”);
    /* Si el archivo no se puede abrir se chequea el valor que
       devuelve fopen */
    if(fp==NULL)
    {
        /* Si no se pudo abrir se tomara la acción adecuada */
        printf(“\n\nError de apertura\n\n”);
        exit(1);
    }
    /* Si se abrió correctamente se procede a cerrar el archivo */
    fclose(fp);
    return 0;
}
//--------------------------------------------------------------------
```

En este ejemplo podemos observar que si el archivo no existía se crea, esto se puede ver mediante el listado del directorio del disco.
Si hubiese existido el archivo destruirá su contenido, podemos observarlo mediante la utilización de un editor de texto, con el que podremos encontrar que el archivo existe pero sin ningún contenido.

Cierre de un archivo

Para cerrar un archivo se utiliza la función fclose.

Su prototipo es:

int fclose(FILE *fp)

Esta función recibe un único parámetro que es el puntero a la estructura FILE que es la que vincula al archivo con el programa.
Devuelve un valor entero que será cero si el cierre del archivo se efectuó correctamente, si ocurre un error devolverá el valor EOF que es el valor -1, esta constante está definida en stdio.h.
Hay que tener en cuenta que siempre se debe cerrar un archivo antes de la finalización del programa, ya que si no se hiciese, no se volcaría el contenido del buffer del stream, perdiéndose los datos que están almacenados en este buffer. También hay que tener en cuenta que pueden ocurrir errores en las entradas que utiliza el sistema operativo para ordenar a los archivos en el disco.

Escritura de un carácter

Cuando se quiere escribir un carácter en un archivo se utiliza la función fputc.

Su prototipo es:

int fputc(int ch,FILE *fp)

Esta función recibe dos parámetros estos son el carácter (ch) que se quiere escribir y un puntero (fp) que asocia a la función con el archivo que se quiere utilizar.
Esta función devuelve un valor entero que indica si se escribió correctamente en el archivo o no. Si se lo hizo en forma exitosa la función devuelve el carácter que se escribió, pero si hubo un error devuelve EOF que es una constante definida en stdio.h y que tiene el valor -1 pero en formato entero.
La función devuelve un int debido a que los distintos caracteres que se pueden escribir en un archivo son 256, tamaño adecuado para una variable de tipo char, pero como también cuando se produce un error devuelve un valor, la cantidad total de datos que se pueden utilizar es de 257 con lo que se necesita más de un byte para poder almacenar todas las combinaciones posibles y por ese motivo se devuelve un tipo int.
Se puede observar que en la función el carácter que se quiere escribir tiene el tipo int, esto se debe a que se desea mantener la concordancia con el tipo devuelto.

Ejemplo:

```
//------------------------------------------------------------------
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
//------------------------------------------------------------------
int main(void)
{
    /* Declara un puntero a FILE para poder manejar el archivo */
    FILE *fp;
    char c=0;
    /* Invoca a la función fopen para abrir el archivo de texto
       nuevoarc.txt en modo de escritura */
    fp=fopen("nuevoarc.txt","w");
    /* Si el archivo no se puede abrir se chequea el valor que
       devuelve fopen */
    if(fp==NULL)
    {
        /* Si no se pudo abrir se tomara la acción adecuada */
        printf("\n\nError de apertura\n\n");
        exit(1);
    }
    /* Si se abrió correctamente se procede a escribir un conjunto de
       caracteres en el archivo */
    /* Se ingresa al lazo de escritura hasta que se ingresa el
       carácter # */
    while(c!='#')
    {
        c=getche();
```

```
        /* Se escribe el carácter en el archivo */
        fputc(c,fp);
    }
    /* Se procede a cerrar el archivo */
    fclose(fp);
    return 0;
}
//-------------------------------------------------------------------
```

Podemos observar el proceso que se debe utilizar para poder escribir caracteres en un archivo determinado.
En este ejemplo nos encontramos que se va a escribir el carácter # que se utiliza para finalizar el lazo de ingreso de datos. Este procedimiento no es recomendable porque se escribe en el archivo un carácter que no pertenece a los datos que se quieren ingresar. Para solucionarlo se deberá modificar el lazo para que esto no suceda.

Lectura de un carácter

Para poder leer un carácter desde un archivo se utiliza la función fgetc.

Su prototipo es:

int fgetc(FILE *fp)

Esta función recibe un puntero a FILE (fp) que es el que permite acceder al archivo.
Devuelve el dato que se ha leído del archivo, este es un carácter pero el tipo que devuelve es int esto se explica porque la función también indica si se produjo un error en la lectura lo indicara, es decir necesita un total de 257 combinaciones, 256 para el carácter leído y uno para indicar el error, como un dato de tipo char solo permite 256 combinaciones posibles, se debe utilizar un int.
El valor devuelto por la función es el carácter leído del archivo cuando la lectura ocurrió correctamente cuando ocurre un error esta función devuelve el valor EOF.
Cada vez que se utiliza una función de lectura o escritura el cursor del archivo se desplaza la cantidad de bytes que se leyeron o escribieron automáticamente.

Ejemplo:

```
//-------------------------------------------------------------------
#include <stdio.h>
#include <stdlib.h>
//-------------------------------------------------------------------
int main(void)
{
        /* Declara un puntero a FILE para poder manejar el archivo */
        FILE *fp;
        char c;
        /* Invoca a la función fopen para abrir el archivo de texto
          nuevoarc.txt en modo de lectura */
        fp=fopen("nuevoarc.txt","r");
        /* Si el archivo no se puede abrir se chequea el valor que
          devuelve fopen */
        if(fp==NULL)
```

```
	{
		/* Si no se pudo abrir se tomara la acción adecuada */
		printf("\n\nError de apertura\n\n");
		exit(1);
	}
	/* Si se abrió correctamente se procede a leer un conjunto de
	   caracteres del archivo */
	/* Se ingresa al lazo de lectura hasta que se lee el carácter # */
	while(c!='#')
	{
		/* Se lee el carácter del archivo */
		c=fgetc(fp);
		/* Se muestra el carácter en pantalla */
		putchar(c);
	}
	/* Se procede a cerrar el archivo */
	fclose(fp);
	return 0;
}
//-----------------------------------------------------------------------
```

Encontramos que en este ejemplo se lee el archivo anteriormente escrito, con el inconveniente que también se leerá el carácter # que no se quiere dentro del grupo de datos.

Detección de fin de archivo

Siempre que se quiere leer un archivo se debe hacerlo hasta que este finalice, esto se logra buscando le señal de fin de archivo (EOF) esta señal no está contenida en el archivo sino en la zona del disco en donde el sistema operativo guarda la ubicación de los distintos sectores de este.
Para poder lograr encontrar el EOF se utiliza la función feof.

Su prototipo es:

int feof(FILE *fp)

Esta función recibe como parámetro un puntero (fp) que es el que la relaciona con el archivo. Devuelve un dato de tipo int que significa falso (0) si no encontró el EOF, ahora si lo encuentra devolverá verdadero (distinto de 0).
Esta función actúa luego de producirse una operación de lectura.

Ejemplo:

```
//-----------------------------------------------------------------------
#include <stdio.h>
#include <stdlib.h>
//-----------------------------------------------------------------------
int main(void)
{
    /* Declara un puntero a FILE para poder manejar el archivo */
    FILE *fp;
    char c;
```

```
    /* Invoca a la función fopen para abrir el archivo de texto
      nuevoarc.txt en modo de lectura */
    fp=fopen("nuevoarc.txt","r");
    /* Si el archivo no se puede abrir se chequea el valor que
      devuelve fopen */
    if(fp==NULL)
    {
       /* Si no se pudo abrir se tomara la acción adecuada */
       printf("\n\nError de apertura\n\n");
       exit(1);
    }
    /* Si se abrió correctamente se procede a leer un conjunto de
      caracteres del archivo */
    /* Se ingresa al lazo de lectura hasta que se encuentra el EOF */
    do
    {
       /* Se lee el carácter del archivo */
       c=fgetc(fp);
       /* Se muestra el carácter en pantalla */
       putchar(c);
    }
    /* Busca la condición de salida del lazo con la detección de EOF */
    while(!feof(fp));
    /* Se procede a cerrar el archivo */
    fclose(fp);
    return 0;
}
//-----------------------------------------------------------------
```

En este ejemplo observamos el uso de la detección del fin de archivo para poder finalizar la lectura del archivo.

Ejemplo:

```
//-----------------------------------------------------------------
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
//-----------------------------------------------------------------
int main(void)
{
    /* Declara dos punteros a FILE para poder manejar a los archivos */
    FILE *in,*out;
    char c;
    /* Invoca a la función fopen para abrir el archivo de texto
      Archiv1.c en modo de lectura */
    /* Si el archivo no se puede abrir se chequea el valor que
      devuelve fopen */
    if(!(in=fopen("Archiv1.c","r")))
    {
       /* Si no se pudo abrir se tomara la acción adecuada */
```

```
        printf("\n\nError de apertura\n\n");
        exit(1);
    }
    /* Invoca a la función fopen para abrir el archivo de texto
       Arc.c en modo de lectura */
    /* Si el archivo no se puede abrir se chequea el valor que
       devuelve fopen */
    if(!(out=fopen("Arc.c","w")))
    {
        /* Si no se pudo abrir se tomara la acción adecuada */
        printf("\n\nError de apertura\n\n");
        exit(1);
    }
    printf("\n\n");
    /* Si se abrieron correctamente se procede copiar el contenido de
       un archivo en el otro */
    /* Se ingresa al lazo hasta que se encuentra el EOF */
    while(!feof(in))
    {
        /* Se lee el carácter del archivo */
        c=fgetc(in);
        /* Se escribe el carácter en el archivo */
        fputc(c,out);
        /* Se muestra el carácter en pantalla */
        printf("%c",c);
    }
    /* Se procede a cerrar los archivos */
    fclose(in);
    fclose(out);
    return 0;
}
//--------------------------------------------------------------------
```

Se muestra cómo se puede copiar un archivo en otro, en este caso se lo hace carácter a carácter.

Detección de error

Luego de una operación realizada en un archivo se puede generar una condición de error que se puede detectar con la función ferror.

Su prototipo es:

int ferror(FILE *fp)

Esta función recibe un puntero (fp) que es el que la relaciona con el archivo.
Devuelve un valor entero que significa verdadero si se produjo una condición de error (distinto de 0) y falso si no ocurrió dicha condición (0).

Lectura de un string

Para poder leer un string se utiliza la función fgets.

Su prototipo es:

char *fgets(char *str,int num, FILE *fp)
recibe tres parámetros un puntero a char (str) que es donde se va a ubicar la cadena de caracteres que se quieren leer, uno entero (num) que es la cantidad de caracteres que se quieren leer y un puntero (fp) que es el que relaciona a la función con el archivo que se quiere utilizar. El segundo parámetro (num) es para que no se exceda el tamaño del vector en donde se va a ubicar el string. Va a leer la cantidad de caracteres que se especificó siempre y cuando no se encuentre un carácter '\n' o el fin de archivo (EOF), si alguna de estas condiciones se produjese leería los caracteres que se encontraran hasta que se hubiese producido alguna de esas condiciones.
Esta función devuelve un puntero a char que es la dirección de comienzo del string, si la operación fue exitosa; en caso contrario, se devuelve un puntero NULL.

Escritura de un string

En el caso de querer escribir un string se utiliza la función fputs.

Su prototipo es:

int fputs(char *str,FILE * fp)

Recibe dos parámetros un puntero a char (str) que es la dirección de comienzo del string que se quiere escribir en el archivo y un puntero a FILE (fp) que es el que la asocia al archivo que se está utilizando.
Devuelve un valor entero que es distinto de cero si la función fue exitosa, si se produjo un error devuelve la constante EOF.
Esta función no escribe en el archivo el fin del string lo que genera un inconveniente para su uso, este inconveniente puede ser solucionado con la escritura de un carácter '\n' cada vez que se escribe un string para indicar su finalización.

Ejemplo:

```
//------------------------------------------------------------------
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
//------------------------------------------------------------------
int main(void)
{
    /* Declara un puntero a FILE para poder manejar el archivo */
    FILE *fp;
    char str[50];
    /* Invoca a la función fopen para abrir el archivo de texto
       arcstr.txt en modo de escritura */
    /* Si el archivo no se puede abrir se chequea el valor que
       devuelve fopen */
    if(!(fp=fopen("arcstr.txt","w")))
    {
        /* Si no se pudo abrir se tomara la acción adecuada */
        printf("\n\nError de apertura\n\n");
        exit(1);
    }
```

```
    printf("\nESCRITURA DEL ARCHIVO\n\n");
    gets(str);
    /* Si se abrió correctamente se procede a escribir un conjunto de
      strings al archivo */
    /* Se ingresa al lazo de escritura hasta que se ingresa el
      string FIN */
    while(strcmp(str,"FIN"))
    {
        /* Se escribe el string en el archivo */
        fputs(str,fp);
        gets(str);
    }
    /* Se procede a cerrar el archivo */
    fclose(fp);
    /* Invoca a la función fopen para abrir el archivo de texto
      arcstr.txt en modo de lectura */
    /* Si el archivo no se puede abrir se chequea el valor que
      devuelve fopen */
    if(!(fp=fopen("arcstr.txt","r")))
    {
        /* Si no se pudo abrir se tomara la acción adecuada */
        printf("\n\nError de apertura\n\n");
        exit(1);
    }
    printf("\nLECTURA DEL ARCHIVO\n\n");
    /* Si se abrió correctamente se procede a leer un conjunto de
      string del archivo */
    /* Se ingresa al lazo de lectura hasta que se encuentre el EOF */
    while(!feof(fp))
    {
        /* Se lee un string de 5 caracteres de largo */
        fgets(str,5,fp);
        /* Se muestra el string en pantalla */
        puts(str);
    }
    /* Se procede a cerrar el archivo */
    fclose(fp);
    return 0;
}
//-------------------------------------------------------------------
```

En este ejemplo se observa cómo se pueden escribir strings y leerlos con un largo especificado.

Ejemplo:

```
//-------------------------------------------------------------------
#include <stdio.h>
#include <stdlib.h>
//-------------------------------------------------------------------
int main(void)
{
```

```
    /* Declara un puntero a FILE para poder manejar el archivo */
    FILE *fp;
    char str[50];
    int largo;
    scanf("%d",&largo);
    /* Invoca a la función fopen para abrir el archivo de texto
      arcstr.txt en modo de escritura */
    /* Si el archivo no se puede abrir se chequea el valor que
      devuelve fopen */
    if(!(fp=fopen("arcstr.txt","r")))
    {
       /* Si no se pudo abrir se tomara la acción adecuada */
       printf("\n\nError de apertura\n\n");
       exit(1);
}
    printf("\nLECTURA DEL ARCHIVO\n\n");
    /* Si se abrió correctamente se procede a leer un conjunto de
      string del archivo */
    /* Se ingresa al lazo de lectura hasta que se encuentre el EOF */
    while(!feof(fp))
    {
       /* Se lee un string del largo especificado en la
         variable */
       fgets(str,largo,fp);
       /* Se muestra el string en pantalla */
       puts(str);
    }
    /* Se procede a cerrar el archivo */
    fclose(fp);
    return 0;
}
//-----------------------------------------------------------------
```

Aquí podemos determinar cómo actúa el tamaño definido en el parámetro de fgets para determinar el largo del string a leer.

Ejemplo:

```
//-----------------------------------------------------------------
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
//-----------------------------------------------------------------
int main(void)
{
    /* Declara un puntero a FILE para poder manejar el archivo */
    FILE *fp;
    char str[50];
    /* Invoca a la función fopen para abrir el archivo de texto
      arcstr.txt en modo de escritura */
    /* Si el archivo no se puede abrir se chequea el valor que
      devuelve fopen */
```

```
    if(!(fp=fopen("arcstr1.txt","w")))
    {
        /* Si no se pudo abrir se tomara la acción adecuada */
        printf("\n\nError de apertura\n\n");
        exit(1);
    }
    printf("\nESCRITURA DEL ARCHIVO\n\n");
    gets(str);
    /* Si se abrió correctamente se procede a escribir un conjunto de
      strings al archivo */
    /* Se ingresa al lazo de escritura hasta que se ingresa el
      string FIN */
    while(strcmp(str,"FIN"))
    {
        /* Se escribe el string en el archivo */
        fputs(str,fp);
        gets(str);
        /* Se escribe el carácter '\n' para guardar la finalización
          del string en el archivo */
        fputc('\n',fp);
    }
    /* Se procede a cerrar el archivo */
    fclose(fp);
    /* Invoca a la función fopen para abrir el archivo de texto
      arcstr.txt en modo de lectura */
    /* Si el archivo no se puede abrir se chequea el valor que
      devuelve fopen */
    if(!(fp=fopen("arcstr1.txt","r")))
    {
        /* Si no se pudo abrir se tomara la acción adecuada */
        printf("\n\nError de apertura\n\n");
        exit(1);
    }
    printf("\nLECTURA DEL ARCHIVO\n\n");
    /* Si se abrió correctamente se procede a leer un conjunto de
      string del archivo */
    /* Se ingresa al lazo de lectura hasta que se encuentre el EOF */
    while(!feof(fp))
    {
        /* Se lee un string de 50 caracteres de largo */
        fgets(str,50,fp);
        /* Se muestra el string en pantalla */
        puts(str);
    }
    /* Se procede a cerrar el archivo */
    fclose(fp);
    return 0;
}
//-------------------------------------------------------------------
```

En este ejemplo aparece una forma de poder corregir la lectura de los string para que correspondan a los que se habían escrito en el archivo.

Entrada con formato

Cuando se quiere ingresar distintos tipos de datos en un archivo se puede utilizar la función fscanf.

Su prototipo es:

int fscanf(FILE *fp, char *control,lista_de_variables);

Esta función recibe tres parámetros un puntero a FILE (fp) que es el que apunta al stream que se quiere utilizar, un puntero a char (control) que es un string que determina el formato de las variables que se quieren leer, tiene la misma forma que se utiliza en scanf y por ultimo un conjunto de direcciones de variables que son las variables en donde se almacenaran los datos (lista_de_variables).
Esta función devuelve la cantidad de ítems leídos si no ocurrió ningún error en la lectura, si hubo algún tipo de error devolverá la constante EOF.
Esta función es similar a la función scanf que se utiliza para ingresar datos a través del teclado, es más la función scanf es un caso particular de fscanf, ya que toma el stream stdin por omisión.

Salida con formato

Cuando se queire escribir datos con formato en un archivo se utiliza la función fprintf.

Su prototipo es:

int fprintf(FILE *fp, char *control, lista_de_variables);

Recibe tres parámetros un puntero a FILE (fp) que es el que apunta al stream utilizado, un puntero a char (control) que es el string que determina el formato de las variables a escribir, tiene la misma forma que se utiliza en printf y por ultimo un conjunto de variables que son los datos que se quieren escribir en el archivo (lista_de_variables).
Esta función devuelve la cantidad de ítems escritos en el caso que la escritura fuese correcta, si se produjo algún error devolverá la constante EOF.
Esta función es similar a la función printf que se utiliza para escribir datos en el monitor, es más la función printf es un caso particular de fprintf, ya que utiliza stdout como stream para enviar los datos.

Ejemplo:

```
//--------------------------------------------------------------------
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
//--------------------------------------------------------------------
int main(void)
{
    /* Declara un puntero a FILE para poder manejar el archivo */
    FILE *fp;
    char str[50],c='x';
    int i,j;
    float f=1.1;
    strcpy(str,"PRUEBA");
    /* Invoca a la función fopen para abrir el archivo de texto
```

```
  Arch1.txt en modo de escritura */
/* Si el archivo no se puede abrir se chequea el valor que
  devuelve fopen */
if(!(fp=fopen("arch1.txt","w")))
{
   /* Si no se pudo abrir se tomara la acción adecuada */
   printf("\n\nError de apertura\n\n");
   exit(1);
}
printf("\nESCRITURA DEL ARCHIVO\n\n");
/* Si se abrió correctamente se procede a escribir los datos
  al archivo */
/* Se escribe un string en el archivo */
fprintf(fp,"%s",str);
printf("%s\n",str);
/* Se escriben 10 datos enteros en el archivo */
for(i=0;i<10;i++)
{
   fprintf(fp,"%d",i);
   printf("%d\n",i);
}
/* Se escribe un carácter en el archivo */
fprintf(fp,"%c",c);
printf("%c\n",c);
/* Se escriben 10 datos flotantes en el archivo */
for(i=0;i<10;i++)
{
   fprintf(fp,"%f",f);
   printf("%f\n",f);
   f*=1.1;
}
/* Se procede a cerrar el archivo */
fclose(fp);
/* Invoca a la función fopen para abrir el archivo de texto
  Arch1.txt en modo de lectura */
/* Si el archivo no se puede abrir se chequea el valor que
  devuelve fopen */
if(!(fp=fopen("arch1.txt","r")))
{
   /* Si no se pudo abrir se tomara la acción adecuada */
   printf("\n\nError de apertura\n\n");
   exit(1);
}
printf("\nLECTURA DEL ARCHIVO\n\n");
/* Si se abrió correctamente se procede a leer un conjunto de
  datos desde el archivo */
/* Se lee un string del archivo */
fscanf(fp,"%s",str);
puts(str);
printf("\n");
/* Se leen 10 datos enteros del archivo */
```

```
    for(j=0;j<10;j++)
    {
        fscanf(fp,"%d",&i);
        printf("%d",i);
    }
    printf("\n");
    /* Se lee un carácter del archivo */
    fscanf("%c",&c);
    printf("%c",c);
    printf("\n");
    /* Se leen 10 datos flotantes del archivo */
    for(i=0;i<10;i++)
    {
        fscanf(fp,"%f",&f);
        printf("%f",f);
    }
    /* Se procede a cerrar el archivo */
    fclose(fp);
    return 0;
}
//-----------------------------------------------------------------
```

En este ejemplo se encuentra que se produce un error en la lectura, ya que cuando se lee es string todos los datos almacenados en el archivo. Como son caracteres ASCII, pueden formar parte del string. Esto sucede porque no se separan los datos en la cadena de control utilizando caracteres de separación entre los datos.
No importa el tipo de dato que se quiera escribir; si no se lo separa con el siguiente, siempre producirá un error en la lectura.

Ejemplo:

```
//-----------------------------------------------------------------
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
//-----------------------------------------------------------------
int main(void)
{
    /* Declara un puntero a FILE para poder manejar el archivo */
    FILE *fp;
    char str[50],c='x';
    int i,j;
    float f=1.1;
    strcpy(str,"PRUEBA");
    /* Invoca a la función fopen para abrir el archivo de texto
      Arch2.txt en modo de escritura */
    /* Si el archivo no se puede abrir se chequea el valor que
      devuelve fopen */
    if(!(fp=fopen("arch2.txt","w")))
    {
        /* Si no se pudo abrir se tomara la acción adecuada */
        printf("\n\nError de apertura\n\n");
```

```
    exit(1);
}
printf("\nESCRITURA DEL ARCHIVO\n\n");
/* Si se abrió correctamente se procede a escribir los datos
   al archivo */
/* Se escribe un string en el archivo */
/* En la cadena de control se agrega el carácter \n para finalizar
   el ingreso del string */
fprintf(fp,"%s\n",str);
printf("%s\n",str);
/* Se escriben 10 datos enteros separador por comas (,)
   en el archivo */
for(i=0;i<10;i++)
{
    fprintf(fp,"%d,",i);
    printf("%d\n",i);
}
/* Se escribe un carácter con una coma como separador */
fprintf(fp,"%c,",c);
printf("%c\n",c);
/* Se escriben 10 datos flotantes separados por comas
   en el archivo */
for(i=0;i<10;i++)
{
    fprintf(fp,"%f,",f);
    printf("%f\n",f);
    f*=1.1;
}
/* Se procede a cerrar el archivo */
fclose(fp);
/* Invoca a la función fopen para abrir el archivo de texto
   Arch2.txt en modo de lectura */
/* Si el archivo no se puede abrir se chequea el valor que
   devuelve fopen */
if(!(fp=fopen("arch2.txt","r")))
{
    /* Si no se pudo abrir se tomara la acción adecuada */
    printf("\n\nError de apertura\n\n");
    exit(1);
}
printf("\nLECTURA DEL ARCHIVO\n\n");
/* Si se abrió correctamente se procede a leer un conjunto de
   daros desde el archivo */
/* Se lee un string y el separador \n del archivo */
fscanf(fp,"%s\n",str);
puts(str);
printf("\n");
/* Se leen 10 datos enteros separados por comas del archivo */
for(j=0;j<10;j++)
{
    fscanf(fp,"%d,",&i);
```

```
        printf("%d\n",i);
    }
    printf("\n");
    /* Se lee un carácter y el separador (,) del archivo */
    fscanf(fp,"%c,",&c);
    printf("%c",c);
    printf("\n");
    /* Se leen 10 datos flotantes separados por comas del archivo */
    for(i=0;i<10;i++)
    {
        fscanf(fp,"%f,",&f);
        printf("%f\n",f);
    }
    /* Se procede a cerrar el archivo */
    fclose(fp);
    return 0;
}
//-------------------------------------------------------------------
```

En este ejemplo se utilizan separadores para poder leer los datos en forma correcta y que no se mezcle un dato con otro.

Lectura y escritura de un bloque de memoria

Cuando se quiere acceder a un archivo de tipo binario se lo debe hacer mediante dos funciones que son: fwrite y fread.
La función fread permite leer datos y fwrite escribirlos. Estos datos, sin importar el tipo de la variable con la que se quiere trabajar, están guardados en el archivo de la misma forma que en la memoria de la computadora. Este es el motivo por el cual se transfiere un bloque de memoria en la operación, tanto de lectura como de escritura.

Función fread

Permite leer un bloque de memoria.

Su prototipo es:
unsigned int fread(void *buf,unsigned int tam,unsigned int cant,FILE *fp);

El primer parámetro que recibe (buf) es un puntero a la dirección de memoria en donde se va a ubicar el dato leído, esta función permite acceder a cualquier tipo de dato, por lo que no se conoce el tipo de dato que tendrá el buffer por lo que el tipo asociado al puntero es void, un puntero genérico.
El siguiente parámetro es (tam) un valor entero no signado que significa el tamaño del tipo de dato en bytes que se va a utilizar.
El parámetro cant se refiere a cuantos bloques de memoria se van a leer, esto permite leer desde el archivo un vector y llenarlo en una sola operación.
El ultimo (fp) es un puntero a FILE que es la dirección del stream que maneja al archivo.
El valor de retorno es la cantidad de bloques leídos si la operación fue correcta, en el caso que se haya producido un error el valor va a ser menor que la cantidad de bloques que se le indico leer.

Función fwrite

Permite escribir un bloque de memoria.
Su prototipo es:
unsigned int fwrite(void *buf,unsigned int tam,unsigned int cant,FILE *fp);

El parámetro buf es un puntero a la dirección de memoria de donde se va a tomar el dato que se quiere escribir, como no se conoce el tipo de dato a escribir es un puntero genérico.
El parámetro tam es un valor entero no signado que indica el tamaño del tipo de dato en bytes que se va a utilizar.
El parámetro cant es la cantidad de bloques de memoria se van a escribir, esto permite escribir un vector en una sola operación en el archivo.
El ultimo (fp) es un puntero a FILE que es la dirección del stream que maneja al archivo.
El valor de retorno es la cantidad de bloques escritos si la operación fue correcta, en el caso que se haya producido un error el valor va a ser menor que la cantidad de bloques que se le indico escribir.

Ejemplo:

```
//------------------------------------------------------------------
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
//------------------------------------------------------------------
struct x
{
    int i;
    float f;
    char c,str[10];
};
//------------------------------------------------------------------
int main(void)
{
    /* Declara un puntero a FILE para poder manejar el archivo */
    FILE *fp;
    char str[50],c='x',str1[50],c1;
    int i,j,k[10];
    float f=1.1,g[10];
    struct x st,st1;
    st.i=11;
    st.f=3.14159;
    st.c='J';
    strcpy(st.str,"prueba");
    strcpy(str,"PRUEBA");
    /* Invoca a la función fopen para abrir el archivo binario
       arch.bin en modo de escritura */
    /* Si el archivo no se puede abrir se chequea el valor que
       devuelve fopen */
    if(!(fp=fopen("arch.bin","wb")))
    {
        /* Si no se pudo abrir se tomara la acción adecuada */
        printf("\n\nError de apertura\n\n");
        exit(1);
```

```
}
/* Si se abrió correctamente se procede a escribir un conjunto de
   datos al archivo */
printf("\nESCRITURA DEL ARCHIVO\n\n");
/* Se escribe un string en el archivo */
fwrite(str,sizeof(str),1,fp);
printf("%s\n",str);
/* Se escriben 10 datos enteros en el archivo */
for(i=0;i<10;i++)
{
    fwrite(&i,sizeof(i),1,fp);
    printf("%d\n",i);
}
/* Se escribe un carácter en el archivo */
fwrite(&c,sizeof(char),1,fp);
printf("%c\n",c);
/* Se escriben 10 datos flotantes en el archivo */
for(i=0;i<10;i++)
{
    fwrite(&f,sizeof(float),1,fp);
    printf("%f\n",f);
    f*=1.1;
}
/* Se escribe una estructura en el archivo */
fwrite(&st,sizeof(struct x),1,fp);
printf("\n%c,%d,%f,%s\n",st.c,st.i,st.f,st.str);
/* Se procede a cerrar el archivo */
fclose(fp);
/* Invoca a la función fopen para abrir el archivo binario
   arch.bin en modo de lectura */
/* Si el archivo no se puede abrir se chequea el valor que
   devuelve fopen */
if(!(fp=fopen("arch.bin","rb")))
{
    /* Si no se pudo abrir se tomara la acción adecuada */
    printf("\n\nError de apertura\n\n");
    exit(1);
}
printf("\nLECTURA DEL ARCHIVO\n\n");
/* Si se abrió correctamente se procede a leer un conjunto de
   datos desde el archivo */
/* Se lee un string del archivo */
fread(str1,sizeof(str1),1,fp);
puts(str1);
printf("\n");
/* Se leen 10 datos enteros del archivo */
fread(k,sizeof(int),10,fp);
for(j=0;j<10;j++)
    printf("%d\n",k[j]);
printf("\n");
/* Se lee un carácter del archivo */
fread(&c1,sizeof(char),1,fp);
```

```
    printf("%c",c1);
    printf("\n");
    /* Se leen 10 datos flotantes del archivo */
    fread(g,sizeof(float),10,fp);
    for(i=0;i<10;i++)
        printf("%f\n",g[i]);
    /* Se lee una estructura del archivo */
    fread(&st1,sizeof(struct x),1,fp);
    printf("\n%c,%d,%f,%s\n",st1.c,st1.i,st1.f,st1.str);
    /* Se procede a cerrar el archivo */
    fclose(fp);
    return 0;
}
//-------------------------------------------------------------------
```

Este ejemplo permite ver cómo se pueden escribir distintos tipos de datos en un archivo binario. Si se compara con el ejemplo que se utilizó con las funciones fscanf y fprintf, se observa claramente que no es necesario separar los datos para que se escriban o lean correctamente, ya que en las operaciones de acceso al archivo se indica la cantidad de bytes que se van a transferir. Por lo que no hay que agregarle separadores entre los datos para que puedan ser diferenciados.

Archivos random o aleatorios

Hasta ahora hemos manejado a los archivos en forma secuencial, es decir leyendo o escribiendo los datos desde el primero al último, esta forma de manejo tiene la desventaja que cuando se quiere acceder a un dato, hay que acceder a todos los anteriores. Esta forma genera un tiempo de acceso que crece a medida que el dato se aleja del principio del archivo, siendo muy variable el tiempo de acceso al archivo.
Existe otra forma de manejo que es el modo aleatorio o random, esta forma de acceso permite poder captar cualquier dato sin tener que recorrer el archivo desde el comienzo al final.
Para este manejo existen funciones de búsqueda y de ubicación dentro del archivo, que permiten poder acceder al lugar del archivo que uno desee sin tener que recorrerlo desde el principio. Estas funciones son fseek y ftell.

Función fseek

Esta función permite ubicar al cursor del archivo en la posición que uno desee.

Su prototipo es:

int fseek(FILE *fp, long desplazamiento, int origen);

Recibe tres parámetros, el primero es un puntero a FILE (fp) que es el que indica en que archivo se va a desplazar, el segundo es un dato de tipo long (desplazamiento) que indica que cuantos bytes se va a desplazar el cursor y el ultimo es el que indica desde donde se va a producir el desplazamiento que es un dato de tipo int (origen) que está representado por tres constantes definidas en stdio.h, estas son: SEEK_SET, SEEK_CUR y SEEK_END.
SEEK_SET indica que se tomara como origen el comienzo del archivo, SEEK_CUR indica que el comienzo de la búsqueda será la posición actual del cursor y SEEK_END el fin del archivo.
Esta función devuelve un valor entero que toma el valor 0 si no se produjo un error y distinto de cero si hubo un error.

Hay que tener en cuenta que el desplazamiento puede ser hacia el fin o hacia el comienzo del archivo, si se quiere avanzzar hacia el fin del archivo este valor sera positivo, pero si se lo desea hacer hacia el comienzo debera tomar un valor negativo.
Otro tema que hay que tener en cuenta que no se puede desplazar antes del comienzo del archivo, ya que esto generaría un error, pero si se puede desplazar más allá del final sin que se produzca un error.
También hay que tener en cuenta que luego de una operación de lectura o escritura el cursor se mueve hacia adelante la misma cantidad de bytes que se leyeron o escribieron. Si se quiere acceder al dato que se accedió, se debe retroceder la cantidad de bytes que se utilizó.

Función ftell

Esta función se utiliza para indicar cuál es la posición del cursor del archivo.

Su prototipo es:

long ftell(FILE *fp);

Recibe un único parámetro que es un puntero a FILE (fp) que se asocia al archivo.
Devuelve un dato de tipo long que indica la posición actual del cursor en bytes desde el comienzo del archivo. Si se produjo un error devuelve el valor -1 en tamaño long.

Retorno al comienzo

La función rewind se utiliza para poder retornar al comienzo del archivo.

Su prototipo es:

void rewind(FILE *fp);

Recibe un puntero (fp) al stream asociado al archivo en el que se quiere llevar el cursor al comienzo del mismo.
No devuelve ningún valor, ya que siempre se puede retornar al comienzo de un archivo, aunque este esté ubicado en el comienzo.

Ejemplo:

```
//---------------------------------------------------------------------
#include <stdio.h>
#include <stdlib.h>
//---------------------------------------------------------------------
int main(void)
{
    /* Declara un puntero a FILE para poder manejar el archivo */
    FILE *in;
    char c;
    long l;
    /* Invoca a la función fopen para abrir el archivo de texto
       Archiv1.c en modo de lectura */
    /* Si el archivo no se puede abrir se chequea el valor que
       devuelve fopen */
    if(!(in=fopen("Archiv1.c","r")))
    {
```

```
    /* Si no se pudo abrir se tomara la acción adecuada */
    printf("\n\nError de apertura\n\n");
    exit(1);
}
printf("\n\n");
/* En la variable l se guarda la posición del cursor y luego se
   la muestra en pantalla */
l=ftell(in);
printf("%ld",l);
fseek(in,0,SEEK_END);
printf("\n\n");
/* En la variable l se guarda la posición del cursor y luego se
   la muestra en pantalla */
/* En este caso se está indicando el tamaño del archivo */
l=ftell(in);
printf("%ld",l);
/* Se lleva el cursor al comienzo del archivo */
rewind(in);
/* Se lee un carácter */
c=fgetc(in);
printf("\n%c",c);
/* Se desplaza 200 bytes desde el comienzo del archivo */
fseek(in,200,SEEK_SET);
/* Se lee un carácter */
c=fgetc(in);
printf("\n%c",c);
/* Se desplaza 220 bytes desde el comienzo del archivo */
fseek(in,220,SEEK_SET);
/* Se lee un carácter */
c=fgetc(in);
printf("\n%c",c);
/* Se desplaza 1 byte desde la posición actual del archivo */
fseek(in,1,SEEK_CUR);
/* Se lee un carácter */
c=fgetc(in);
printf("\n%c",c);
/* Se desplaza 0 bytes desde la posición actual del archivo */
fseek(in,0,SEEK_CUR);
/* Se lee un carácter */
c=fgetc(in);
printf("\n%c",c);
/* Se desplaza 0 bytes desde la posición actual del archivo */
fseek(in,0,SEEK_CUR);
/* Se lee un carácter */
c=fgetc(in);
printf("\n%c",c);
/* Se desplaza -1 byte desde la posición actual del archivo,
   un byte hacia el comienzo */
fseek(in,-1,SEEK_CUR);
/* Se lee un carácter */
c=fgetc(in);
```

```
    printf("\n%c",c);
    /* Se desplaza 200 bytes desde el comienzo del archivo */
    fseek(in,200,SEEK_SET);
    /* Se lee un carácter */
    c=fgetc(in);
    printf("\n%c",c);
    /* Se desplaza 200 bytes desde el comienzo del archivo */
    fseek(in,200,SEEK_SET);
    /* Se lee un carácter */
    c=fgetc(in);
    printf("\n%c",c);
    /* Se desplaza -150 bytes desde el fin del archivo */
    fseek(in,-150,SEEK_END);
    /* Se lee un carácter */
    c=fgetc(in);
    printf("\n%c",c);
    /* Se procede a cerrar el archivo */
    fclose(in);
    return 0;
}
//-------------------------------------------------------------------
```

En este programa de ejemplo se puede ver cómo se puede acceder en forma aleatoria a un archivo.

Borrar un archivo

Existe una función que permite borrar un archivo, sin necesidad de recurrir a un stream que se denomina remove.

Su prototipo es:

int remove(char *nombre);

Esta recibe el nombre del archivo, hay que tener en cuenta que lo que se denomina nombre en un archivo es un término que consta de cuatro partes: unidad, ruta, nombre y extensión. El valor de retorno de esta función es un entero que toma el valor 0 si la función se ejecuto correctamente, en cambio si se produjo una condición de error el valor es distinto de cero.

Liberar el buffer

Cuando se quiere liberar el contenido del buffer del stream se utiliza la función fflush.

Su prototipo es:

int fflush(FILE *fp);

Recibe el puntero al stream (fp) que asocia a la función con el stream que se quiere liberar. Retorna un valor entero que toma el valor 0 si se ejecutó sin error, en caso contrario devuelve la constante EOF.
Hay que tener en cuenta que el contenido del buffer se pierde, en ningún caso se resguardan los datos en el archivo.

Altas, bajas y modificaciones

Una de las aplicaciones más comunes en el uso de archivos es lo que se denomina ABM (altas, bajas y modificaciones).
Este tipo de utilización consiste en manejar un conjunto de información y poder guardarla en un archivo para que sea procesada.
La información que se puede almacenar es de cualquier tipo, datos de personas, características de productos, lista de artículos que se comercializan en un establecimiento, etc.
Podemos dividirlo en tres partes:

- Altas
- Bajas
- Modificaciones

Altas

Significa que esta parte permite ingresar nuevos valores para que sean almacenados en el archivo.

Bajas

Permite eliminar algunos registros del archivo.

Modificaciones

Esta sección del programa permite modificar los contenidos de los registros seleccionados.

Ejemplo:

En el programa que sigue muestra cómo se puede manejar una agenda telefónica. En este también se puede buscar un determinado dato, y permite mostrar un listado completo de los datos almacenados.

```
//-----------------------------------------------------------------
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <string.h>
//-----------------------------------------------------------------
struct Direccion
{
    char Calle[20];
    int Numero;
    char Localidad[20],Provincia[20];
};
//-----------------------------------------------------------------
struct Dato
{
```

```
    char Nombre[20],Apellido[20];
    struct Direccion Casa;
    int TelefonoFijo;
    int Celular;
};
//------------------------------------------------------------------
char Menu(void);
void Alta(void);
struct Dato Ingresar(void);
void Ordenar(void);
int Compara(struct Dato,struct Dato);
void Mostrar(struct Dato);
void Bajar(void);
long Buscar(char *,char *,FILE *,FILE *);
void Listar(void);
void Modificar(void);
void Encontrar(void);
void Compactar(void);
void IngresaNyA(char *,char *);
void DatoNoEncontrado(char *,char *);
//------------------------------------------------------------------
int main(void)
{
    char Opcion;
    do
    {
        /* Invoca a la función Menu() para que escriba el menu de
          opciones */
        Opcion=Menu();
        switch(Opcion)
        {
        case '1':
            /* Invoca a la función Alta() para que maneje el ingreso de
              los datos */
            Alta();
            break;
        case '2':
            /* Invoca a la función Bajar() para que maneje el borrado de
              los datos */
            Bajar();
            break;
        case '3':
```

```
                /* Invoca a la función Modificar() para que maneje la
                   modificacion de los datos */
                Modificar();
                break;
            case '4':
                /* Invoca a la función Encontrar() para que maneje la
                   busqueda de los datos */
                Encontrar();
                break;
            case '5':
                /* Invoca a la función Listar() para que maneje el
                   listado de los datos */
                Listar();
                break;
            }
    }
    while(Opcion!='6');
    /* Invoca a la función Compactar() para que elimine de los archivos
       los datos eliminados */
    Compactar();
    return 0;
}
//-------------------------------------------------------------------
char Menu(void)
{
    /* Esta función genera el menú de acceso a las diferentes opciones
       y devuelve la opción elegida */
    system("cls");
    printf("\n\n1 - Ingresar un dato");
    printf("\n\n2 - Borrar un dato");
    printf("\n\n3 - Modificar un dato");
    printf("\n\n4 - Buscar un dato");
    printf("\n\n5 - Listar");
    printf("\n\n6 - Salir");
    printf("\n\n\n\n Ingrese una Opción ( 1 - 6 ) ");
    return getche();
}
//-------------------------------------------------------------------
void Alta(void)
{
    /* Esta función maneja el ingreso de datos en el archivo agenda.dat
```

```
    y la generación de los índices en el archivo índice.dat */
  /* Declara dos punteros a FILE para poder manejar a los archivos */
  FILE *Agenda,*Indice;
  struct Dato Contacto;
  long Posicion;
  /* Invoca a la función fopen para abrir el archivo binario
    agenda.dat en modo de lectura extendida */
  /* Si el archivo no se puede abrir se chequea el valor que
    devuelve fopen */
  if(!(Agenda=fopen("agenda.dat","r+b")))
     /* Invoca a la función fopen para abrir el archivo binario
       agenda.dat en modo de escritura */
     Agenda=fopen("agenda.dat","wb");
  /* Invoca a la función fopen para abrir el archivo binario
    indice.dat en modo de lectura extendida */
  /* Si el archivo no se puede abrir se chequea el valor que
    devuelve fopen */
  if(!(Indice=fopen("indice.dat","r+b")))
     /* Invoca a la función fopen para abrir el archivo binario
       indice.dat en modo de escritura */
     Indice=fopen("indice.dat","wb");
  /* Invoca a la función ingresar para que se ingresen los datos
    desde el teclado */
  Contacto=Ingresar();
  /* Se desplaza al fin del archivo agenda.dat */
  fseek(Agenda,0L,SEEK_END);
  /* Se desplaza al fin del archivo indice.dat */
  fseek(Indice,0L,SEEK_END);
  /* Obtiene la posición en el archivo agenda.dat para utilizarlo
    como índice */
  Posicion=ftell(Agenda);
  /* Escribe el dato en el archivo agenda.dat */
  fwrite(&Contacto,sizeof Contacto,1,Agenda);
  /* Escribe la ubicación del dato en el archivo indice.dat */
  fwrite(&Posicion,sizeof Posicion,1,Indice);
  /* Se procede a cerrar los archivos */
  fclose(Agenda);
  fclose(Indice);
  /* Invoca a la función para ordenar el archivo índice.dat */
  Ordenar();
}
//------------------------------------------------------------------
```

```
struct Dato Ingresar(void)
{
    /* Permite el ingreso desde teclado de los datos que se van
       a procesar */
    struct Dato Contacto;
    system("cls");
    printf("\nIngreso de Datos\n\n");
    printf("\nNombre : ");
    gets(Contacto.Nombre);
    printf("\nApellido : ");
    gets(Contacto.Apellido);
    printf("\nTelefono : ");
    scanf("%d",&Contacto.TelefonoFijo);
    printf("\nCelular : ");
    scanf("%d",&Contacto.Celular);
    fflush(stdin);
    printf("\nDirección\n\nCalle : ");
    gets(Contacto.Casa.Calle);
    printf("\nNumero : ");
    scanf("%d",&Contacto.Casa.Numero);
    fflush(stdin);
    printf("\nLocalidad : ");
    gets(Contacto.Casa.Localidad);
    printf("\nProvincia : ");
    gets(Contacto.Casa.Provincia);
    return Contacto;
}
//--------------------------------------------------------------------
void Ordenar(void)
{
    /* Realiza el ordenamiento de los datos mediante la modificación
       del orden de los índices en el archivo indice.dat */
    /* Declara dos punteros a FILE para poder manejar a los archivos */
    FILE *Agenda,*Indice;
    long Desplazamiento,Posicion;
    struct Dato Minimo,Auxiliar;
    int Total,i,j,Ubicacion;
    /* Invoca a la función fopen para abrir el archivo binario
       agenda.dat en modo de lectura */
    /* Si el archivo no se puede abrir se chequea el valor que
     devuelve fopen */
   if(!(Agenda=fopen("agenda.dat","rb")))
```

```
{
    /* Si no se pudo abrir se tomara la acción adecuada */
    printf("\n\nError de apertura”);
    printf("\n\nPresione una tecla para continuar");
    getch();
    return;
}
/* Invoca a la función fopen para abrir el archivo binario
   indice.dat en modo de lectura extendida */
/* Si el archivo no se puede abrir se chequea el valor que
   devuelve fopen */
if(!(Indice=fopen("indice.dat","r+b")))
{
    /* Si no se pudo abrir se tomara la acción adecuada */
    printf("\n\nError de apertura”);
    printf("\n\nPresione una tecla para continuar");
    getch();
    return;
}
/* Se desplaza al fin del archivo indice.dat */
fseek(Indice,0L,SEEK_END);
/* Obtiene la cantidad de registros del archivo */
Total=ftell(Indice)/sizeof(long);
/* Se desplaza al principio del archivo indice.dat */
fseek(Indice,0L,SEEK_SET);
/* Se utiliza el método de la selección para ordenar el archivo
   de índices */
for(i=0;i<Total-1;i++)
{
    Ubicacion=i;
    /* Se desplaza la cantidad de registros determinado por Ubicación
       en el archivo indice.dat */
    fseek(Indice,(long)(Ubicacion*sizeof(long)),SEEK_SET);
    /* Obtiene la ubicación del dato */
    fread(&Posicion,sizeof(long),1,Indice);
    /* Se desplaza la cantidad de registros determinado por Posicion
       en el archivo agenda.dat */
    fseek(Agenda,Posicion,SEEK_SET);
    /* Obtiene el dato */
    fread(&Minimo,sizeof(struct Dato),1,Agenda);
    for(j=i+1;j<Total;j++)
    {
```

```
            /* Se desplaza la cantidad de registros determinado por
              j en el archivo indice.dat */
            fseek(Indice,(long)(j*sizeof(long)),SEEK_SET);
            /* Obtiene la ubicación del dato */
            fread(&Desplazamiento,sizeof(long),1,Indice);
            /* Se desplaza la cantidad de registros determinado por
              Desplazamiento en el archivo agenda.dat */
            fseek(Agenda,Desplazamiento,SEEK_SET);
            /* Obtiene el dato */
            fread(&Auxiliar,sizeof(struct Dato),1,Agenda);
            if(Compara(Minimo,Auxiliar))
            {
                Minimo=Auxiliar;
                Posicion=Desplazamiento;
                Ubicacion=j;
            }
        }
        /* Se desplaza la cantidad de registros determinado por i
          en el archivo indice.dat */
        fseek(Indice,(long)(i*sizeof(long)),SEEK_SET);
        /* Obtiene la ubicación del dato */
        fread(&Desplazamiento,sizeof(long),1,Indice);
        /* Se desplaza la cantidad de registros determinado por Ubicación
          en el archivo indice.dat */
        fseek(Indice,(long)(Ubicacion*sizeof(long)),SEEK_SET);
        /* Escribe la ubicación del dato en el archivo indice.dat */
        fwrite(&Desplazamiento,sizeof(long),1,Indice);
        /* Se desplaza la cantidad de registros determinado por i
          en el archivo indice.dat */
        /* Se desplaza la cantidad de registros determinado por i
           en el archivo indice.dat */
        fseek(Indice,(long)(i*sizeof(long)),SEEK_SET);
        /* Escribe la ubicación del dato en el archivo indice.dat */
        fwrite(&Posicion,sizeof(long),1,Indice);
    }
    /* Se procede a cerrar los archivos */
    fclose(Agenda);
    fclose(Indice);
}
//-----------------------------------------------------------------
int Compara(struct Dato Minimo,struct Dato Auxiliar)
{
```

```
    /* Funcion que compara dos strings y devuelve 1 si uno es menor
      que el otro */
    int valor;
    valor=strcmp(Minimo.Apellido,Auxiliar.Apellido);
    if(valor>0)
        return 1;
    if(!valor)
    {
        valor=strcmp(Minimo.Nombre,Auxiliar.Nombre);
        if(valor>0)
            return 1;
    }
    return 0;
}
//-----------------------------------------------------------------
void Mostrar(struct Dato Elemento)
{
    /* Muestra en pantalla los datos */
    system("cls");
    printf("\n\n\n\nNombre : ");
    puts(Elemento.Nombre);
    printf("\nApellido : ");
    puts(Elemento.Apellido);
    printf("\nTelefono : %d",Elemento.TelefonoFijo);
    printf("\n\nCelular : %d",Elemento.Celular);
    printf("\n\nDirección\n\nCalle : ");
    puts(Elemento.Casa.Calle);
    printf("\nNumero : %d",Elemento.Casa.Numero);
    printf("\n\nLocalidad : ");
    puts(Elemento.Casa.Localidad);
    printf("\nProvincia : ");
    puts(Elemento.Casa.Provincia);
}
//-----------------------------------------------------------------
void Bajar(void)
{
    /* Funcion para borrar datos del archivo */
    /* Declara dos punteros a FILE para poder manejar a los archivos */
    FILE *Agenda,*Indice;
    struct Dato Elemento;
    char Nombre[20],Apellido[20],Opcion;
    long Posicion;
```

```
/* Invoca a la función fopen para abrir el archivo binario
   agenda.dat en modo de lectura extendida */
/* Si el archivo no se puede abrir se chequea el valor que
   devuelve fopen */
if(!(Agenda=fopen("agenda.dat","r+b")))
{
     /* Si no se pudo abrir se tomara la acción adecuada */
     printf("\n\nError de apertura”);
     printf("\n\nPresione una tecla para continuar");
     getch();
     return;
}
/* Invoca a la función fopen para abrir el archivo binario
   indice.dat en modo de lectura */
/* Si el archivo no se puede abrir se chequea el valor que
   devuelve fopen */
if(!(Indice=fopen("indice.dat","rb")))
{
     /* Si no se pudo abrir se tomara la acción adecuada */
     printf("\n\nError de apertura”);
     printf("\n\nPresione una tecla para continuar");
     getch();
     return;
}
/* Invoca a la función IngresaNyA para que se ingrese el dato
   a buscar */
IngresaNyA(Apellido,Nombre);
/* Invoca a la función Buscar para que encuentre la posición del
   dato que se quiere eliminar */
Posicion=Buscar(Apellido,Nombre,Agenda,Indice);
/* Si el Dato no fue encontrado avisa y continua */
if(Posicion==-1)
{
     DatoNoEncontrado(Apellido,Nombre);
     printf("\n\nPresione una tecla para continuar");
     getch();
}
else
{
     /* Se encontró el dato */
    /* Se desplaza la cantidad de registros determinado por Posicion
       en el archivo agenda.dat */
```

```
        fseek(Agenda,Posicion,SEEK_SET);
        /* Obtiene el dato */
        fread(&Elemento,sizeof Elemento,1,Agenda);
        /* Invoca a la función Mostrar para que el dato se
           muestre en pantalla */
        Mostrar(Elemento);
        /* Pide la confirmación del borrado */
        printf("\n\n\nDesea eliminarlo (s/n)? ");
        Opcion=getche();
        if(Opcion=='s'||Opcion=='S')
        {
            /* Le asigna a Nombre y Apellido un string nulo para
               que lo detecte como borrado */
            strcpy(Elemento.Nombre,"");
            strcpy(Elemento.Apellido,"");
            /* Se desplaza la cantidad de registros determinado por
               Posicion en el archivo agenda.dat */
            fseek(Agenda,Posicion,SEEK_SET);
            /* Escribe el dato en el archivo agenda.dat */
            fwrite(&Elemento,sizeof Elemento,1,Agenda);
        }
    }
    /* Se procede a cerrar los archivos */
    fclose(Agenda);
    fclose(Indice);
}
//-----------------------------------------------------------------------
long Buscar(char *Apellido,char *Nombre,FILE *Agenda,FILE *Indice)
{
    /* Funcion que realiza una búsqueda binaria en el archivo */
    int Medio,Izquierda,Derecha;
    long Ubicacion;
    struct Dato Contacto;
    /* Se desplaza al fin del archivo indice.dat */
    fseek(Indice,0L,SEEK_END);
    /* Obtiene la cantidad de registros del archivo */
    /* En derecha y en izauierda guarda los limites para realizar
       la búsqueda */
    Derecha=ftell(Indice)/sizeof(long);
    Izquierda=0;
    /* Recorre el archivo mientras no se invierten los limites */
    while(Izquierda<=Derecha)
```

```
    {
        /* Realiza la búsqueda en el medio del archivo */
        Medio=(Izquierda+Derecha)/2;
        /* Se desplaza la cantidad de registros determinado por Medio
           en el archivo indice.dat */
        fseek(Indice,(long)(Medio*sizeof(long)),SEEK_SET);
        /* Obtiene la ubicación del dato */
        fread(&Ubicacion,sizeof(long),1,Indice);
        /* Se desplaza la cantidad de registros determinado por Ubicacion
           en el archivo agenda.dat */
        fseek(Agenda,Ubicacion,SEEK_SET);
        /* Obtiene el dato */
        fread(&Contacto,sizeof(struct Dato),1,Agenda);
        /* Encontro el dato y devuelve su ubicación */
        if(!strcmp(Apellido,Contacto.Apellido)&&
           !strcmp(Nombre,Contacto.Nombre))
            return Ubicacion;
        /* Determina la zona en donde continuar la búsqueda */
        if((strcmp(Apellido,Contacto.Apellido)>0)||
          ((!strcmp(Apellido,Contacto.Apellido))&&
          (strcmp(Nombre,Contacto.Nombre)<0)))
            Izquierda=Medio+1;
        else
            Derecha=Medio-1;
    }
    /* Devuelve -1 porque no encontró el dato */
    return -1L;
}
//-------------------------------------------------------------------
void Listar(void)
{
    /* Funcion que realiza el listado del archivo en pantalla */
    /* Declara dos punteros a FILE para poder manejar a los archivos */
    FILE *Agenda,*Indice;
    struct Dato Elemento;
    int Total,i;
    long Posicion;
    /* Invoca a la función fopen para abrir el archivo binario
       agenda.dat en modo de lectura */
    /* Si el archivo no se puede abrir se chequea el valor que
       devuelve fopen */
    if(!(Agenda=fopen("agenda.dat","rb")))
```

```
{
    /* Si no se pudo abrir se tomara la acción adecuada */
    printf("\n\nError de apertura”);
    printf("\n\nPresione una tecla para continuar");
    getch();
    return;
}
/* Invoca a la función fopen para abrir el archivo binario
   indice.dat en modo de lectura */
/* Si el archivo no se puede abrir se chequea el valor que
   devuelve fopen */
if(!(Indice=fopen("indice.dat","rb")))
{
    /* Si no se pudo abrir se tomara la acción adecuada */
    printf("\n\nError de apertura”);
    printf("\n\nPresione una tecla para continuar");
    getch();
    return;
}
/* Se desplaza al fin del archivo indice.dat */
fseek(Indice,0L,SEEK_END);
/* Obtiene la cantidad de registros del archivo */
Total=ftell(Indice)/sizeof(long);
/* Se desplaza al principio del archivo indice.dat */
fseek(Indice,0L,SEEK_SET);
/* Recorre el archivo leyendo los datos a mostrar */
for(i=0;i<Total;i++)
{
    /* Se desplaza la cantidad de registros determinado por i
       en el archivo indice.dat */
    fseek(Indice,(long)(i*sizeof(long)),SEEK_SET);
    /* Obtiene la ubicación del dato */
    fread(&Posicion,sizeof(long),1,Indice);
    /* Se desplaza la cantidad de registros determinado por Posicion
       en el archivo agenda.dat */
    fseek(Agenda,Posicion,SEEK_SET);
    /* Obtiene el dato */
    fread(&Elemento,sizeof(struct Dato),1,Agenda);
    /* Verifica que el dato no haya sido borrado */
    if(strcmp(Elemento.Apellido,""))
    {
        /* Invoca a la función Mostrar para que el dato se
```

```
              muestre en pantalla */
            Mostrar(Elemento);
            printf("\n\n\nPresione una tecla para continuar ");
            getch();
        }
    }
    /* Se procede a cerrar los archivos */
    fclose(Agenda);
    fclose(Indice);
}
//-----------------------------------------------------------------
void Modificar(void)
{
    /* Funcion que permite modificar datos en el archivo */
    /* Declara dos punteros a FILE para poder manejar a los archivos */
    FILE *Agenda,*Indice;
    struct Dato Elemento;
    char Nombre[20],Apellido[20],Opcion;
    long Posicion;
    /* Invoca a la función fopen para abrir el archivo binario
       agenda.dat en modo de lectura extendida */
    /* Si el archivo no se puede abrir se chequea el valor que
       devuelve fopen */
    if(!(Agenda=fopen("agenda.dat","r+b")))
    {
        /* Si no se pudo abrir se tomara la acción adecuada */
        printf("\n\nError de apertura");
        printf("\n\nPresione una tecla para continuar");
        getch();
        return;
    }
    /* Invoca a la función fopen para abrir el archivo binario
       indice.dat en modo de lectura extendida */
    /* Si el archivo no se puede abrir se chequea el valor que
       devuelve fopen */
    if(!(Indice=fopen("indice.dat","r+b")))
    {
        /* Si no se pudo abrir se tomara la acción adecuada */
        printf("\n\nError de apertura");
        printf("\n\nPresione una tecla para continuar");
        getch();
        return;
```

```
}
/* Invoca a la función IngresaNyA para que se ingrese el dato
   a modificar */
IngresaNyA(Apellido,Nombre);
/* Invoca a la función Buscar para que encuentre la posición del
   dato que se quiere modificar */
Posicion=Buscar(Apellido,Nombre,Agenda,Indice);
/* Si el Dato no fue encontrado avisa y continua */
if(Posicion==-1)
{
    DatoNoEncontrado(Apellido,Nombre);
    printf("\n\nPresione una tecla para continuar");
    getch();
}
else
{
    /* Se encontró el dato */
    /* Se desplaza la cantidad de registros determinado por Posicion
       en el archivo agenda.dat */
    fseek(Agenda,Posicion,SEEK_SET);
    /* Obtiene el dato */
    fread(&Elemento,sizeof Elemento,1,Agenda);
    /* Invoca a la función Mostrar para que el dato se
       muestre en pantalla */
    Mostrar(Elemento);
    printf("\n\n\nDesea modificarlo (s/n)? ");
    Opcion=getche();
    if(Opcion=='s'||Opcion=='S')
    {
        do
        {
            /* Presenta un menu que presenta las opciones para
               elegir el campo que se quiere modificar */
            system("cls");
            printf("\n\nElija que desea modificar");
            printf("\n\n1 – Nombre");
            printf("\n\n2 – Apellido");
            printf("\n\n3 – Telefono");
            printf("\n\n4 – Celular");
            printf("\n\n5 - Calle");
            printf("\n\n6 – Numero");
            printf("\n\n6 – Localidad");
```

```
printf("\n\n8 - Provincia");
printf("\n\n9 – Terminar");
printf("\n\n\nIngrese una Opción ( 1 - 9 ) ");
Opcion=getche();
switch(Opcion)
{
case '1':
    printf("\n\n\n\n\nNombre : ");
    gets(Elemento.Nombre);
    break;
case '2':
    printf("\n\n\n\n\nApellido : ");
    gets(Elemento.Apellido);
    break;
case '3':
    printf("\n\n\n\n\nTelefono : ");
    scanf("%d",&Elemento.TelefonoFijo);
    fflush(stdin);
    break;
case '4':
    printf("\n\n\n\n\nCelular : ");
    scanf("%d",&Elemento.Celular);
    fflush(stdin);
    break;
case '5':
    printf("\n\n\n\n\nCalle : ");
    gets(Elemento.Casa.Calle);
    break;
case '6':
    printf("\n\n\n\n\nNumero : ");
    scanf("%d",&Elemento.Casa.Numero);
    fflush(stdin);
    break;
case '7':
    printf("\n\n\n\n\nLocalidad : ");
    gets(Elemento.Casa.Localidad);
    break;
case '8':
    printf("\n\n\n\n\nProvincia : ");
    gets(Elemento.Casa.Provincia);
    break;
}
```

```
            }
            while(Opcion!='9');
        }
        /* Se desplaza la cantidad de registros determinado por Posicion
           en el archivo agenda.dat */
        fseek(Agenda,Posicion,SEEK_SET);
        /* Escribe el dato en el archivo agenda.dat */
        fwrite(&Elemento,sizeof Elemento,1,Agenda);
    }
    /* Se procede a cerrar los archivos */
    fclose(Agenda);
    fclose(Indice);
    /* Invoca a la función para ordenar el archivo índice.dat */
    Ordenar();
}
//------------------------------------------------------------------
void Encontrar(void)
{
    /* Funcion que permite encontrar un dato dentro del archivo */
    /* Declara dos punteros a FILE para poder manejar a los archivos */
    FILE *Agenda,*Indice;
    struct Dato Elemento;
    char Nombre[20],Apellido[20],Opcion;
    long Posicion;
    /* Invoca a la función fopen para abrir el archivo binario
       agenda.dat en modo de lectura */
    /* Si el archivo no se puede abrir se chequea el valor que
       devuelve fopen */
    if(!(Agenda=fopen("agenda.dat","rb")))
    {
        /* Si no se pudo abrir se tomara la acción adecuada */
        printf("\n\nError de apertura”);
        printf("\n\nPresione una tecla para continuar");
        getch();
        return;
    }
    /* Invoca a la función fopen para abrir el archivo binario
       indice.dat en modo de lectura */
    /* Si el archivo no se puede abrir se chequea el valor que
       devuelve fopen */
    if(!(Indice=fopen("indice.dat","rb")))
    {
```

```
        /* Si no se pudo abrir se tomara la acción adecuada */
        printf("\n\nError de apertura”);
        printf("\n\nPresione una tecla para continuar");
        getch();
        return;
    }
    do
    {
        /* Invoca a la función IngresaNyA para que se ingrese el dato
          a encontrar */
        IngresaNyA(Apellido,Nombre);
        /* Invoca a la función Buscar para que encuentre la posición del
          dato que se quiere encontrar */
        Posicion=Buscar(Apellido,Nombre,Agenda,Indice);
        /* Si el Dato no fue encontrado avisa y continua */
        if(Posicion==-1)
            DatoNoEncontrado(Apellido,Nombre);
        else
        {
            /* Se encontró el dato */
            /* Se desplaza la cantidad de registros determinado por
              Posicion en el archivo agenda.dat */
            fseek(Agenda,Posicion,SEEK_SET);
            /* Obtiene el dato */
            fread(&Elemento,sizeof Elemento,1,Agenda);
            /* Invoca a la función Mostrar para que el dato se
              muestre en pantalla */
            Mostrar(Elemento);
        }
        printf("\n\n\nDesea buscar otro dato (s/n)? ");
        Opcion=getche();
    }
    while(Opcion=='s'||Opcion=='S');
    /* Se procede a cerrar los archivos */
    fclose(Agenda);
    fclose(Indice);
}
//------------------------------------------------------------------
void Compactar(void)
{
    /* Funcion que permite eliminar del archivo los registros
      borrados lógicamente */
```

```
/* Declara tres punteros a FILE para poder manejar a los archivos */
FILE *Agenda,*Indice,*Auxiliar;
struct Dato Elemento;
long Posicion;
int Total,i;
/* Invoca a la función fopen para abrir el archivo binario
  agenda.dat en modo de lectura */
/* Si el archivo no se puede abrir se chequea el valor que
  devuelve fopen */
if(!(Agenda=fopen("agenda.dat","rb")))
    /* Si no se pudo abrir se tomara la acción adecuada */
    return;
/* Invoca a la función fopen para abrir el archivo binario
  tmp.tmp en modo de escritura */
/* Si el archivo no se puede abrir se chequea el valor que
  devuelve fopen */
if(!(Auxiliar=fopen("tmp.tmp","wb")))
    /* Si no se pudo abrir se tomara la acción adecuada */
    return;
/* Invoca a la función fopen para abrir el archivo binario
  indice.dat en modo de escritura */
/* Si el archivo no se puede abrir se chequea el valor que
  devuelve fopen */
if(!(Indice=fopen("indice.dat","wb")))
    /* Si no se pudo abrir se tomara la acción adecuada */
    return;
/* Se desplaza al fin del archivo agenda.dat */
fseek(Agenda,0L,SEEK_END);
/* Obtiene la cantidad de registros del archivo */
Total=ftell(Agenda)/sizeof(struct Dato);
/* Se desplaza al principio del archivo indice.dat */
fseek(Indice,0L,SEEK_SET);
/* Se desplaza al principio del archivo agenda.dat */
fseek(Agenda,0L,SEEK_SET);
/* Se desplaza al principio del archivo tmp.tmp */
fseek(Auxiliar,0L,SEEK_SET);
for(i=0;i<Total;i++)
{
    /* Obtiene el dato */
    fread(&Elemento,sizeof(struct Dato),1,Agenda);
     if(strcmp(Elemento.Apellido,""))
     {
```

```
            /* Obtiene la posición en el archivo agenda.dat para
               utilizarlo como índice */
            Posicion=ftell(Auxiliar);
            /* Escribe la ubicación del dato en el archivo indice.dat */
            fwrite(&Posicion,sizeof(long),1,Indice);
            /* Escribe el dato en el archivo tmp.tmp */
            fwrite(&Elemento,sizeof(struct Dato),1,Auxiliar);
        }
    }
    /* Se procede a cerrar los archivos */
    fclose(Indice);
    fclose(Agenda);
    fclose(Auxiliar);
    /* Borra el archivo agenda.dat */
    remove("agenda.dat");
    /* Renombra el archivo tmp.tmp como agenda.dat */
    system("ren tmp.tmp agenda.dat");
}
//-------------------------------------------------------------------
void IngresaNyA(char *Apellido,char *Nombre)
{
    /* Funcion que permite ingresar Nombre y Apellido */
    system("cls");
    printf("\n\nIngrese el Apellido : ");
    gets(Apellido);
    printf("\n\nIngrese el Nombre : ");
    gets(Nombre);
}
//-------------------------------------------------------------------
void DatoNoEncontrado(char *Apellido,char *Nombre)
{
    /* Funcion que muestra el Nombre y Apellido */
    system("cls");
    printf("\n\n %s, %s",Apellido,Nombre);
    printf("\n\n No ha sido encontrado");
}
//-------------------------------------------------------------------
```

El sistema utiliza dos archivos, uno para el guardado de los datos de la agenda y el otro para guardar la posición de comienzo de cada uno de los datos. Este tipo de archivo se denomina índice, ya que permite tener un índice de búsqueda en los datos.
En la opción de "Ingresar un dato" se agrega al archivo de los datos los valores ingresados por teclado; y en el archivo índice, la posición de comienzo de ese dato.
En "Borrar un dato" se guarda en el archivo de datos el Nombre y Apellido como strings vacíos para generar un borrado lógico.
La opción "Modificar un dato" permite cambiar el valor de cualquier campo de los datos con un nuevo valor y se lo guarda modificado en el archivo de datos.
En "Buscar un dato" sé realiza la búsqueda de un dato en el archivo y luego se lo muestra en la pantalla.
Por ultimo, la opción "Listar" va a mostrar en pantalla los contenidos almacenados en el archivo de datos.
Cuando se finaliza la ejecución del programa, se llama a una función que va a borrar los datos eliminados lógicamente del archivo de datos, haciendo que el tamaño del archivo disminuya, ya que no hay datos que no posean un significado. También se regenera el archivo de índices, ya que los datos que no están presentes en el archivo de datos no ocupan una posición y por ello no hace falta guardarla.
Cuando se realiza un alta o una modificación en el archivo se procede a reordenar los índices. Este proceso se realiza para permitir utilizar una búsqueda binaria cada vez que se desee encontrar un determinado dato.

TEMA 7

VARIABLES DINÁMICAS

En muchas aplicaciones no se conoce con anticipación la cantidad de memoria necesaria para las variables a utilizar en un programa.
Esto ocurre porque no se tiene como conocimiento de todas los datos que se procesarán o también por la posibilidad de poder ubicarlos en diversas posiciones de la memoria para ejecutar mas velozmente dicha aplicación.
Para poder realizar este proceso se utilizan las variables dinámicas.

Asignación dinámica de memoria

El compilador permite poder ubicar variables en la memoria en forma dinámica, es decir, otorgarle a las diferentes variables espacio a medida que se necesita.
Esto se puede realizar ya que el compilador divide a la memoria en cuatro zonas, donde se ubica el código, en donde existen todas las variables globales, el stack (pila) y el heap (zona de memoria libre).
En esta última es donde se puede otorgar memoria para alojar a las variables dinámicas.
El compilador tiene para ello funciones que permiten realizar dichas asignaciones estas son: malloc(),realloc() y free(). Todas estas están ubicadas en el header stdlib.h.

Función malloc()

La función malloc() posibilita asignar memoria libre para la cantidad de espacio requerida.

Su prototipo es :

void *malloc(size_t tamaño_requerido);

El tamaño requerido se expresa en bytes. Si no se conoce con exactitud dicha cantidad se usa la sentencia sizeof, ésta retorna el tamaño en bytes de la expresión que recibe.
La función malloc devuelve un puntero a void que contiene la dirección de memoria en donde ésta ubicado el bloque asignado por ella. En caso de que no exista lugar disponible en la memoria para asignar dicha solicitud retorna un puntero nulo.
Un puntero a void es un puntero que no apunta a ningún tipo específico, éste se incrementa en un byte. Normalmente se debe realizar un casting de dicho puntero.
El tipo size_t es una redefinición del tipo unsigned int.

Ejemplo

```
1.  #include<stdlib.h>
2.  #include<stdio.h>
3.  #include<conio.h>
4.  void main(void)
```

```
5.  {
6.  int *p;
7.  p=(int *)malloc(sizeof(int));
8.  if(!p)
9.  {
10. printf("\nNo hay memoria suficiente");
11. printf("\nPresione una tecla para continuar");
12. getch();
13. exit(1);
14. }
15. printf("\nIngrese un valor entero ");
16. scanf("%d",p);
17. printf("\n\nEl valor ingresado en la dirección %p es %d",p,*p);
18. free(p);
19. }
```

En la línea 7 se asigna el espacio de memoria para un puntero a entero llamado p.
En la 7 se utiliza malloc para reservar el tamaño de un entero en la memoria dinamica, sizeof retorna el tamaño en bytes.
Entre las líneas 8 a 14 se verifica si el valor del puntero retornado por malloc es nulo o no. En el caso de ser nulo se indica que no hay memoria suficiente para la operación y se sale del programa, en caso contrario se ingresa el valor esperado.
En la 17 se imprime tanto la dirección a la que apunta p como el contenido de la posición de memoria apuntada.
En la línea 18 se libera la memoria asignada por malloc().
La salida que se obtiene es la siguiente:

Ingrese un valor entero 23

El valor ingresado en la dirección 07E0 es 23

Sizeof

Devuelve el tamaño en bytes de la expresión o del tipo dado.
Su sintaxis es:

sizeof <expresión>

O

sizeof (<tipo>)

Ejemplo

```
sizeof(int);
```

Devuelve el tamaño del tipo int.

```
int vec[10];
sizeof(vec);
sizeof(vec[4]);
```

Aquí tenemos que el primer sizeof devuelve el tamaño total del vector vec, en cambio en el segundo devolverá el tamaño de un int porque vec[4] es una variable de tipo int.

Función realloc()

Esta función reubica un bloque de memoria pedido. Se utiliza cuando se requiere expandir o contraer un determinado bloque anteriormente pedido,
Su prototipo es :

void *realloc(void *bloque_pedido,size_t tamaño_requerido);

El puntero bloque es el puntero retornado por la función malloc(), que asigno la cantidad inicial de memoria requerida. Tamaño_requerido es la nueva cantidad de memoria requerida.
Retorna un valor de puntero que indica la nueva posición de memoria asignada, en caso que no haya capacidad de memoria retorna un puntero nulo.

Ejemplo

```
char *p;
p=(char *)malloc(10*sizeof(char));
p-(char *)realloc(p,30);
```

Primero se asignan 10 bytes de tipo char en la dirección apuntada por p luego se reubica el bloque inicialmente pedido para agrandar este bloque a 30 bytes en la nueva dirección apuntada por p.

Función free()

Libera un bloque de memoria pedido por las funciones malloc() o calloc().
Su prototipo es :

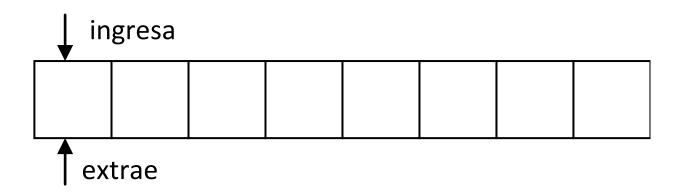
void free(void *bloque_pedido);

TEMA 8

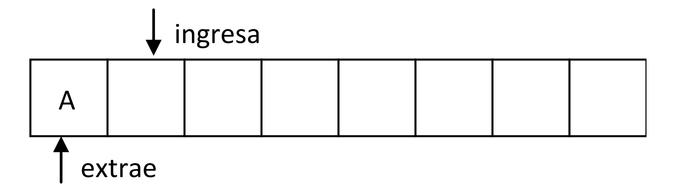
ESTRUCTURA DE DATOS

Cola

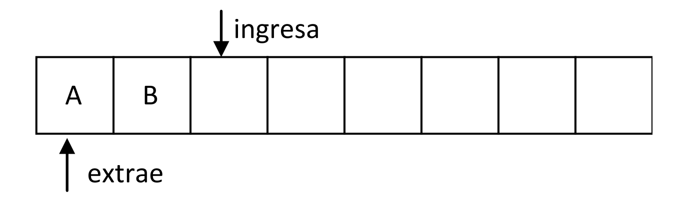
Una cola es una estructura de datos en la cual los datos se ingresan y se extraen únicamente en el orden en que se ingresan. En este tipo de estructura de datos tiene la particularidad que cuando se extrae un dato éste se destruye.
También se las conocen como buffer o memoria de tipo fifo (first input first output), este último nombre se debe a la forma en que se ingresan o se extraen los datos.
Se deben utilizar dos indicadores, uno para conocer el lugar en donde se va a incluir un dato y otro para conocer donde esta ubicado el primer dato que va a ser extraído.
En general podemos decir que este tipo de estructuras se va a almacenar en un lugar limitado de memoria, pero esto no siempre es posible debido a que en algunas circunstancias no se puede conocer la cantidad máxima de datos a utilizar. Este problema se soluciona utilizando una lista, ingresando y extrayendo los datos como si fuese una cola.
Al comenzar a utilizarla, ambos indicadores se apuntan al comienzo de la misma. Como resultado de esto se dice que la cola esta vacía.
En el siguiente gráfico se indica el estado inicial de la cola:

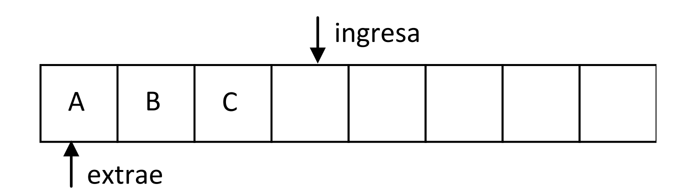


Si se ingresa un dato el índice de ingreso se moverá hacia delante en un lugar, pero el de salida de datos permanecerá fijo. Este efecto se puede observar en el dibujo siguiente.

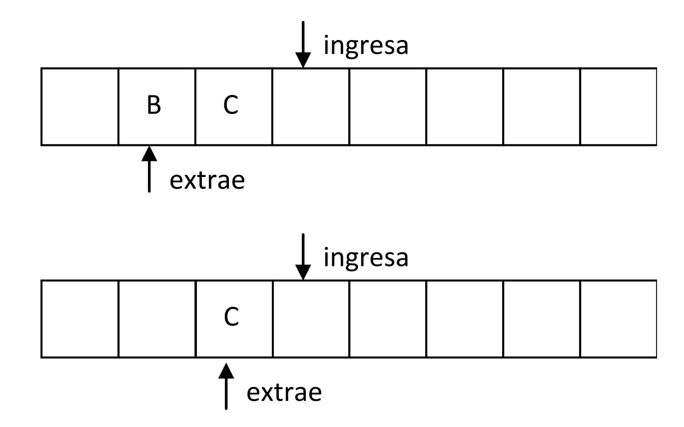


A medida que se ingresan datos podemos observar que solo se desplaza el índice de ingreso.





Luego de ingresar 3 datos. Podemos ver el estado de los índices en el dibujo anterior.
Ahora en los dibujos siguientes observamos como se mueve el índice de salida de datos a medida que se extrae un dato por vez, también se encuentra que el índice de entrada no se desplaza.



De esta forma quedaría la cola luego de extraer dos datos.
La forma en que se produce este proceso ocasiona una disminución del tamaño útil de la misma en la medida en que esta se utiliza.
Esto es una desventaja de este tipo de estructura y este inconveniente se evita usando una cola circular.

Ejemplo

```
1.  //--------------------------------------------------------------------
2.  #include<stdio.h>
3.  #include<stdlib.h>
4.  #include<conio.h>
5.  #include<ctype.h>
6.  //--------------------------------------------------------------------
7.  #define MAX 100
8.  //--------------------------------------------------------------------
9.  void Ingreso(int *,int *,float *);
10. void Lee(int *,int *,float *);
11. //--------------------------------------------------------------------
12. int main(void)
13. {
14. int Ingresa=0,Extrae=0,Opción;
15. float Cola[MAX];
16. do
17. {
18. system(“CLS”);
19. printf(“1 - Ingresar datos\n”);
20. printf(“2 - Leer datos\n”);
21. printf(“3 - Salir\n\n”);
22. printf(“Ingrese una opción ( 1 - 3 ) : “);
```

```
23. scanf("%d",&Opción);
24. switch(Opción)
25. {
26. case 1:
27. Ingreso(&Ingresa,&Extrae,Cola);
28. break;
29. case 2:
30. Lee(&Ingresa,&Extrae,Cola);
31. break;
32. case 3:
33. exit(0);
34. }
35. }
36. while(Opción!=3);
37. return 0;
38. }
39. //--------------------------------------------------------------
40. void Ingreso(int *Escritura,int *Lectura,float *Cola)
41. {
42. if(*Escritura==MAX)
43. {
44. system("CLS");
45. printf("\n\nCola llena");
46. printf("\nPresione una tecla para continuar");
47. getch();
48. return;
49. }
50. printf("\nIngrese el dato : ");
51. scanf("%f",&Cola[*Escritura]);
52. (*Escritura)++;
53. printf("\n\n\nPresione una tecla para continuar");
54. getch();
55. }
56. //--------------------------------------------------------------
57. void Lee(int *Escritura,int *Lectura,float *Cola)
58. {
59. if(*Escritura==*Lectura)
60. {
61. system("CLS");
62. printf("\n\nCola vacia");
63. printf("\nPresione una tecla para continuar");
64. getch();
65. return;
66. }
67. printf("\nEl dato es : %f",Cola[*Lectura]);
68. (*Lectura)++;
69. printf("\n\n\nPresione una tecla para continuar");
70. getch();
71. }
72. //--------------------------------------------------------------
```

En el ejemplo anterior se utiliza una cola para almacenar un conjunto de hasta 100 valores del tipo punto flotante.

Este programa utiliza dos funciones para el manejo de la cola:

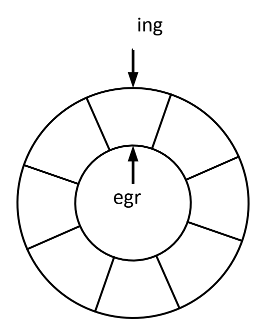
- Ingreso: con esta función se produce el ingreso de los datos en la cola.
- Lee: la función lee extrae valores de la cola.

La función main se define entre las líneas 12 a 38. En ellas se genera un menú de opciones para poder utilizar la cola.
En la línea 14 se declarán las variables Ingresa y Extrae que indicarán los lugares en donde se pueden ingresar o extraer un valor, ambas se inicializan al comienzo de la cola. En la línea 15 se declara un vector de tipo float de 100 posiciones, en éste se van a almacenar los datos que procesa el programa.
Entre las líneas 16 a 36 se inicia un lazo en el cual se genera el menú de opciones utilizando un switch - case para seleccionar la opción elegida. En el caso de elegir la opción "Salir" se utiliza la función exit con un valor 0, que indica que el programa ha terminado normalmente.
Cuando se selecciona la opción "Ingresar datos" se invoca a la función Ingreso, pasándole los parámetros de comienzo y fin de la cola como así también el vector donde se guardan los datos. Estos parámetros se pasan por referencia.
Al seleccionar la opción "Leer datos" se invoca a la función Lee, pasándosele los mismos parámetros que en la función Ingreso.
La función Ingreso recibe dos punteros a int y un puntero a float; El primer puntero (Escritura) contiene la dirección de la posición en donde se ingresará el dato, el segundo (Lectura) contiene la dirección de la posición para poder extraer el primer dato y el último puntero (Cola) contiene la dirección de comienzo del vector de datos. Su valor de retorno es void, está definida entre las líneas 40 a 55.
El contenido de lo apuntado por Escritura (línea 42) se constata que no supere el máximo valor del vector, en caso de hacerlo se imprime un mensaje alusivo, y luego se retorna a la función main; Este proceso se encuentra entre las líneas 42 a 49.
En caso de que se pueda ingresar un valor, esto se realizará en la línea 51. Se incrementa el índice de escritura para dejar a la cola preparada para un nuevo ingreso de datos, finalizando la función.
La función Lee recibe tres punteros, dos int y uno float, estos punteros tienen el mismo significado que para la función Ingreso, y devuelve void. Esta definida entre las líneas 57 a 71.
En la línea 59 se verifica si la posición de, ingreso es la misma que la posición de salida, a través del contenido de lo apuntado por los punteros Escritura y Lectura, en caso de que así lo sean, indicará que la cola esta vacía y retornará a la función main. Este proceso se encuentra entre las líneas 59 a 66. Cuando existe un dato para extraer de la cola lo mostrará en la línea 67 y luego incrementará el índice de extracción para poder extraer un nuevo dato de ser solicitado.

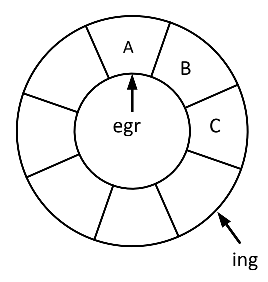
Cola circular

Una cola circular es una estructura de datos derivada de la cola.
La mejora se produce mediante la posibilidad de desplazar los índices al comienzo de la misma siempre y cuando esto sea posible. Es decir, si la cola no está llena se puede seguir introduciendo datos desde el principio; y cuando el indicador de salida llega al final y hay datos que extraer, también se lo lleva al comienzo y en ambos casos se pueden ingresar o extraer datos. Esto hace que el tamaño útil no decaiga.
El proceso de ingreso o egreso de datos es idéntico a la cola, pero la diferencia consiste en la posibilidad de retornar al comienzo de la misma mediante un salto circular.

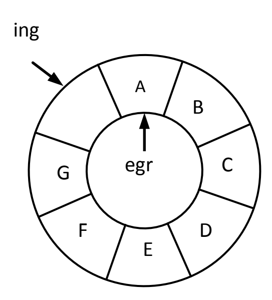
La ubicación inicial de los índices es la que se indica en el dibujo siguiente:



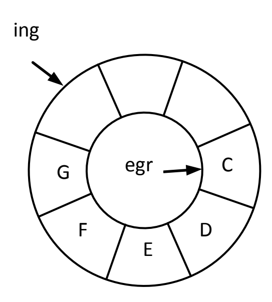
A medida que se vayan ingresando los datos se irá corriendo el índice de ingreso. Por ejemplo, después de haber ingresado tres datos, la posición de los índices sería la siguiente:



Cuando se llena la cola, se encontrarían los índices en las siguientes posiciones:



El inconveniente que presenta este tipo de colas es que se desperdicia una posición para guardar los datos.
En la ilustración siguiente se grafica como queda la cola luego de extraer dos datos, y en esta situación vemos que nuevamente podemos volver a ingresar nuevos datos.



De esta forma se podrían ingresar nuevos datos sin perder tamaño útil en la cola.

Ejemplo

```
1.    //------------------------------------------------------------
2.    #include<stdio.h>
3.    #include<stdlib.h>
4.    #include<conio.h>
5.    #include<ctype.h>
```

```
6.    //----------------------------------------------------------------
7.    #define MAX 100
8.    //----------------------------------------------------------------
9.    void Ingresa(int *,int **,int *,int *,int *);
10. void Lee(int *,int **,int *,int *,int *);
11. //----------------------------------------------------------------
12. int main(void)
13. {
14.   int Llena=0,Vacia=1,Opción,Cola[MAX],*Extrae=Cola,*Ingreso=Cola;
15.   do
16.   {
17.     system("CLS");
18.     printf("1 - Ingresa datos\n");
19.     printf("2 - Lee datos\n");
20.     printf("3 - Salir\n\n");
21.     printf("Ingrese una opción ( 1 - 3 ) : ");
22.     fflush(stdin);
23.     scanf("%d",&Opción);
24.     switch(Opción)
25.     {
26.     case 1:
27.       Ingresa(Cola,&Ingreso,&Llena,&Vacia,Extrae);
28.       break;
29.     case 2:
30.       Lee(Cola,&Extrae,&Vacia,&Llena,Ingreso);
31.       break;
32.     case 3:
33.       exit(0);
34.     }
35.   }
36.   while(Opción!=3);
37.   return 0;
38. }
39. //----------------------------------------------------------------
40. void Ingresa(int *Cola,int **Ingr,int *Llena,int *Vacia,int *Extr)
41. {
42.   if(*Llena)
43.   {
44.     system("CLS");
45.     printf("\n\nCola llena");
46.     printf("\n\nPresione una tecla para continuar\n");
47.     getch();
48.     return;
49.   }
50.   printf("\n\nIngrese el dato : ");
51.   scanf("%d",*Ingr);
52.   *Vacia=0;
53.   (*Ingr)++;
54.   if(*Ingr==Cola+MAX)
55.     *Ingr=Cola;
56.   if(*Ingr==Extr)
```

```
57.     *Llena=1;
58. }
59. //----------------------------------------------------------------
60. void Lee(int *Cola,int **Extr,int *Vacia,int *Llena,int *Ingr)
61. {
62.   if(*Vacia)
63.   {
64.     system("CLS");
65.     printf("\n\nCola vacia");
66.     printf("\n\nPresione una tecla para continuar\n");
67.     getch();
68.     return;
69.   }
70.   *Llena=0;
71.   printf("\n\nEl dato es : %d\n",**Extr);
72.   printf("\n\nPresione una tecla para continuar\n");
73.   getch();
74.   (*Extr)++;
75.   if(*Extr==Cola+MAX)
76.     *Extr=Cola;
77.   if(*Extr==Ingr)
78.     *Vacia=1;
79. }
80. //----------------------------------------------------------------
```

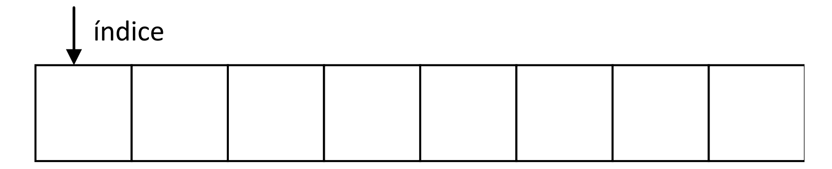
Podemos observar en el ejemplo anterior un programa que permite el ingreso y la extracción de los elementos en una cola circular, esta tiene una capacidad máxima de 100 datos.
Entre las líneas 12 a 38 se define la función main esta permite la elección de cada uno de los ítems anteriormente mencionados mediante el uso de un menú de opciones. En este se llaman a las funciones correspondientes.
Se declara un vector de datos en donde se generará la cola, dos variables (Llena y Vacía) que indican el estado de la cola, y dos punteros (Extrae e Ingreso) que van a apuntar a las posiciones de ingreso y extracción de los datos, iniciándolos con la dirección del vector que indica la posición inicial de la cola.
La función Ingreso permitirá la entrada de los datos en la cola. A esta se le pasan mediante punteros los parámetros que se utilizan para indicar la dirección del vector, el estado de la cola y los índices de ingreso y egreso.
En las líneas 42 a 49 se verifica si la cola esta llena informando dicho acontecimiento y luego retornando a main.
De no cumplirse el hecho de que esté llena, se ingresa el dato (línea 51), luego en la línea 52 se modifica la bandera Vacia, para indicar que la cola no esta vacía y se incrementa la posición de ingreso de datos (línea 53).
En las líneas 54 y 55 se determina si se llegó al máximo del vector y en tal caso se produce un salto circular llevándolo al principio nuevamente.
En las dos líneas siguientes se verifica si los índices de ingreso y extracción son iguales, esto señalaría que la cola se ha llenado modificando la bandera de cola llena.
Desde la línea 60 a 79 se define la función de extracción de datos en la cola. A ella se le pasan los parámetros mediante referencia, estos valores son la dirección del vector de datos, los índices de ingreso y egreso, y las banderas de vacío y lleno.
Entre las líneas 62 a 69 se determina si la cola esta vacía o no, en dicho caso se informará y luego se retorna a main. Si no ocurre esto la bandera Llena se desactiva (línea 70). En la siguiente se imprime el dato en cuestión y luego se incrementa el índice de extracción.

En la 75 y 76 se examina si el índice de extracción llega al máximo valor y en tal situación se produce el salto circular.
En las dos líneas siguientes se establece si los índices coinciden en cuya situación se modifica la bandera Vacia.

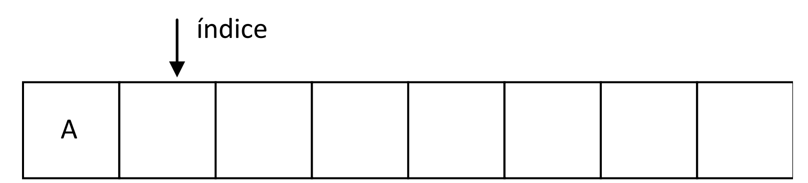
Pila

En este tipo de estructura los datos se extraen en orden inverso al que se ingresan, es decir que el último dato que se introduce es el primero en salir, de allí su denominación de memoria lifo (last input first output) o stack. También en ella al quitar un dato éste se destruye y por lo tanto no es posible volver a utilizarlo.

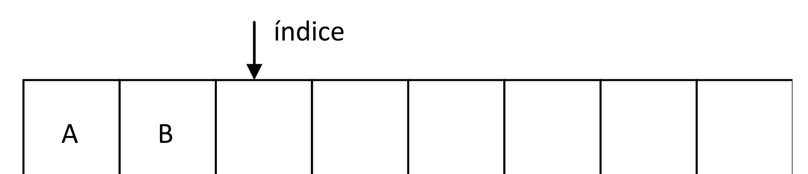
Se deberá utilizar un solo índice que actuará como puntero para poder indicar cuál es el lugar en donde se ingresará o se extraerá el dato.
Este puntero indicará el próximo lugar libre en la memoria para ingresar un dato, cuando se quiere sacar un dato de la pila solamente hay que disminuir este puntero quitando el dato y así poder permitir un nuevo ingreso.
En la figura se aprecia la posición inicial del índice cuando la pila esta vacía.



Cuando se ingresa un dato el índice se moverá en una posición e indicará la próxima ubicación disponible para un dato. Esto puede observarse en la próxima ilustración.



Si se procede a ingresar nuevos datos éstos seguirán apilándose en la memoria y encontraríamos el siguiente estado en la pila.



Luego de ingresar tres datos la pila se verá de la siguiente forma.

índice

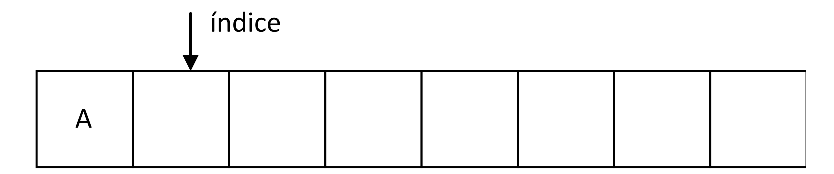
A	B	C					

Luego al sacar un dato quedaría de esta forma.

índice

A	B						

Al quitar un nuevo dato tendríamos lo siguiente.



Como podemos observar la pila aumenta y decrece en tamaño a medida que se ingresan o extraen datos.

Ejemplo

```
1.  //-------------------------------------------------------------
2.  #include<stdio.h>
3.  #include<stdlib.h>
4.  #include<conio.h>
5.  //-------------------------------------------------------------
6.  #define MAX 100
7.  //-------------------------------------------------------------
8.  void Ingresa(int *,float *);
9.  void Lee(int *,float *);
10. //-------------------------------------------------------------
11. int main(void)
12. {
13.   int Indice=0,Opción;
14.   float Pila[MAX];
15.   do
16.   {
17.     system("CLS");
18.     printf("1 - Ingresa datos\n");
19.     printf("2 - Lee datos\n");
20.     printf("3 - Salir\n\n");
21.     printf("Ingrese una opción ( 1 - 3 ) : ");
22.     scanf("%d",&Opción);
23.     switch(Opción)
24.     {
25.     case 1:
26.       Ingresa(&Indice,Pila);
27.       break;
28.     case 2:
29.       Lee(&Indice,Pila);
30.       break;
31.     case 3:
32.       exit(0);
33.     }
34.   }
35.   while(Opción!=3);
36.   return 0;
37. }
38. //-------------------------------------------------------------
39. void Ingresa(int *Indice,float *Pila)
40. {
41.   if(*Indice==MAX)
42.   {
```

```
43.     system("CLS");
44.     printf("\n\nPila llena");
45.     printf("\n\nPresione una tecla para continuar\n");
46.     getch();
47.     return;
48.   }
49.   system("CLS");
50.   printf("\nIngrese el dato : ");
51.   scanf("%f",&Pila[*Indice]);
52.   (*Indice)++;
53. }
54. //-------------------------------------------------------------
55. void Lee(int *Indice,float *Pila)
56. {
57.   if(!*Indice)
58.   {
59.     system("CLS");
60.     printf("\n\nPila vacia");
61.     printf("\n\nPresione una tecla para continuar\n");
62.     getch();
63.     return;
64.   }
65.   (*Indice)--;
66.   printf("\nEl dato es : %f",Pila[*Indice]);
67.   printf("\n\nPresione una tecla para continuar\n");
68.   getch();
69. }
70. //-------------------------------------------------------------
```

Podemos ver en el ejemplo un algoritmo que permite el ingreso y extracción de una estructura del tipo pila.
En la línea 6 se define el valor del tamaño de la pila, y en este caso se le da un valor de 100.
Entre las líneas 11 a 37 se define la función main. En esta se usa un lazo para ingresar a un menú de opciones:

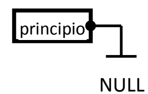
- Ingresa datos
- Lee datos
- Salir

En la línea 13 se declara e inicializa una variable llamada Indice que es el índice de ingreso de la pila. Este toma un valor inicial cero para indicar que la misma está vacía.
A continuación se declara un vector que van a ser los encargados de guardar los datos que se van a ingresar.
Este menú lleva a las funciones correspondientes, cuyos prototipos están en las líneas 8 y 9.
Entre las líneas 39 a 53 se define la función Ingresa, esta recibe un puntero a int llamado Indice que se va encargar del valor de ingreso en la pila. Hay una variable llamada Pila que es un puntero a float a la que se le va a pasar la dirección de comienzo del vector de float.
Si *Indice es igual a MAX quiere decir que la pila esta llena. Dada esta condición se informa al respecto y se retorna a la función main, el proceso se realiza entre las líneas 41 a 48.
En el caso que no se cumpla la condición anterior se ingresa el dato esto se realiza en la línea 51. En la 52 se incrementa el índice de la pila finalizando la función.
Desde la línea 55 a 69 se define la función Lee que será la encargada de extraer y mostrar un dato por vez. Recibe un puntero a int que toma el valor del índice de la pila y un puntero a float que recibe la dirección de comienzo del vector de datos.

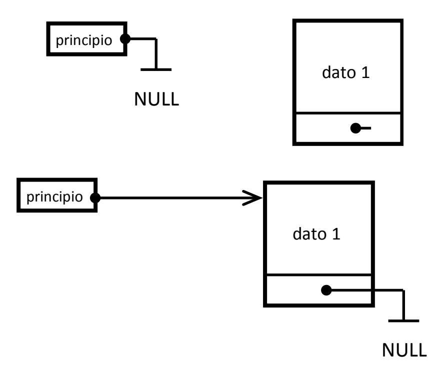
Como primer paso se verifica si la pila no esta vacía, esto se realiza analizando el contenido del índice, si este es cero indicará esta condición, informado tal hecho y luego retornando a main (líneas 57 a 64).
En la línea 65 se decrementa el valor del índice, debido a que éste esta apuntando a la primera posición libre en la pila.
En la línea 66 se muestra el dato y finaliza la función.

Lista simplemente enlazada

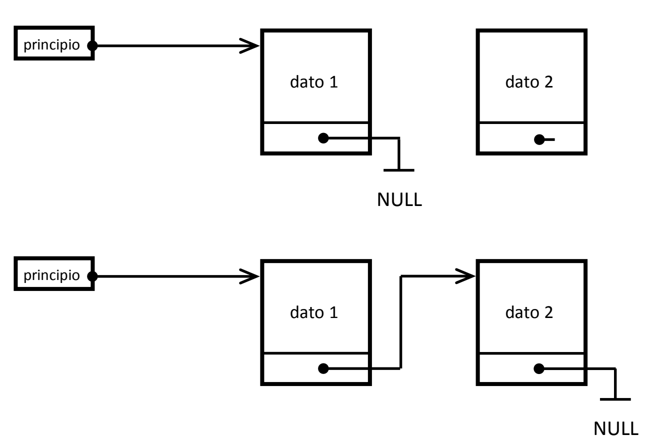
Una lista es un tipo de estructura de datos que forma una cadena mediante un enlace entre un dato y el siguiente. Como existe un enlace entre los diferentes datos no esta acotada la cantidad de posibles datos a ser procesados.
El límite máximo lo impone la memoria, debido a que cada conjunto de datos debe ubicarse en una posición de memoria distinta, esta debe ser solicitada mediante el uso de las funciones de asignación dinámica (por ej. malloc()).
Para poder utilizar este tipo de estructura se deben agrupar los datos con un puntero, esto se realiza mediante el uso de una estructura y dentro de ella un puntero a la misma. Además debemos agregar un indicativo al principio de los datos, esto lo debemos realizar mediante un puntero a estructura.
Antes de comenzar a crear la lista, el puntero que indica el comienzo, debe apuntar a un valor NULL, que indica que la lista esta vacía. Este caso lo vemos en la figura.



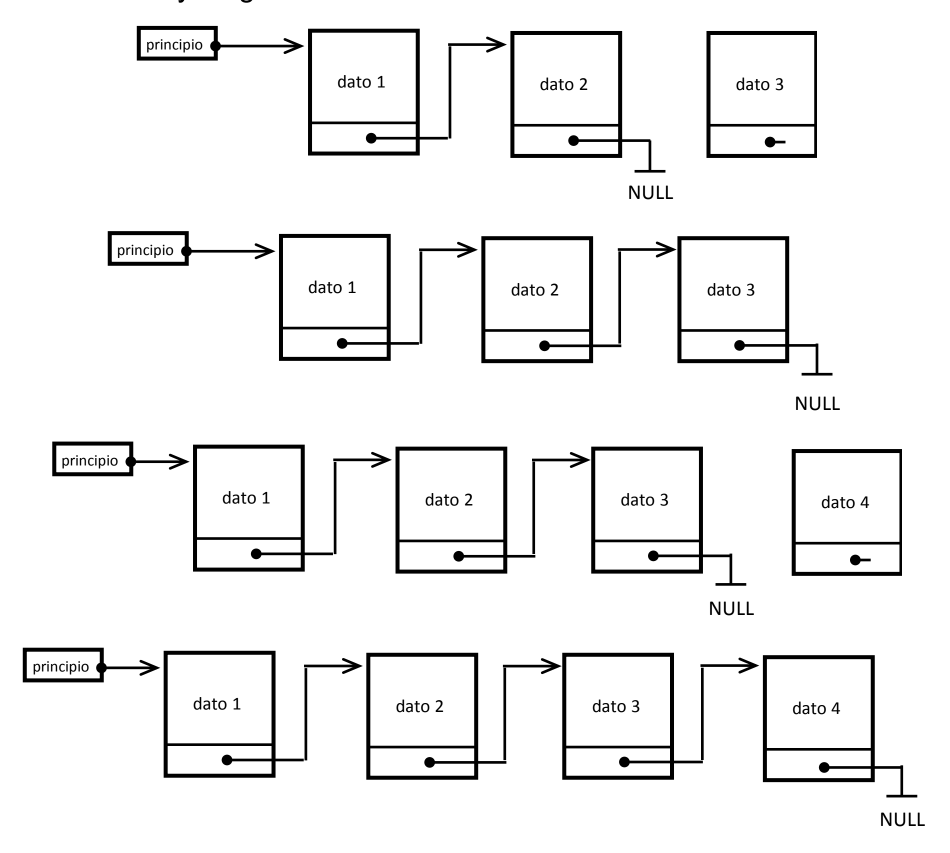
Cuando se genera el primer conjunto de datos el puntero de comienzo debe apuntar a este conjunto y el puntero del dato deberá apuntar a NULL, indicando el fin de la lista. Como podemos visualizar en el siguiente dibujo.



El siguiente elemento puede ubicarse a continuación del último para lo cual el puntero del conjunto de datos que existe en la lista se deberá apuntar a la nueva estructura y el puntero de esta apuntará a NULL.



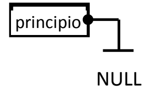
Así sucesivamente pueden incluirse los datos uno por uno, veamos como se irían incluyendo un tercer dato a la lista y luego un cuarto.



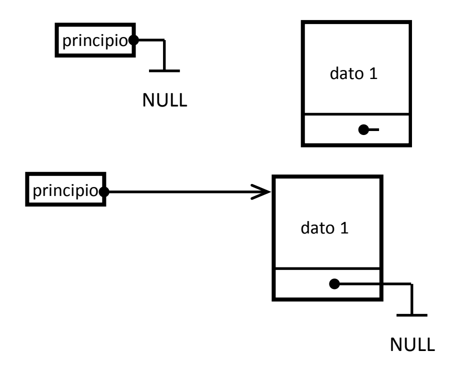
De esta forma podemos ir haciendo crecer la lista a medida que se adicionan los datos, pero de esta manera los datos están desordenados.
La manera vista anteriormente no es la más eficiente para manejar los datos, pero sí es sencilla; una mejor ubicación de los datos sería, ordenarlos según una determinada llave. Este método permite insertar cada conjunto de datos en el lugar adecuado según el orden deseado. Aquí se pueden presentar tres posibilidades:

- Insertar como primer elemento.
- Insertar como último elemento.
- Insertar como un elemento intermedio.

Siempre que la lista este vacía el puntero de comienzo estará apuntando a NULL, como en el caso anterior.



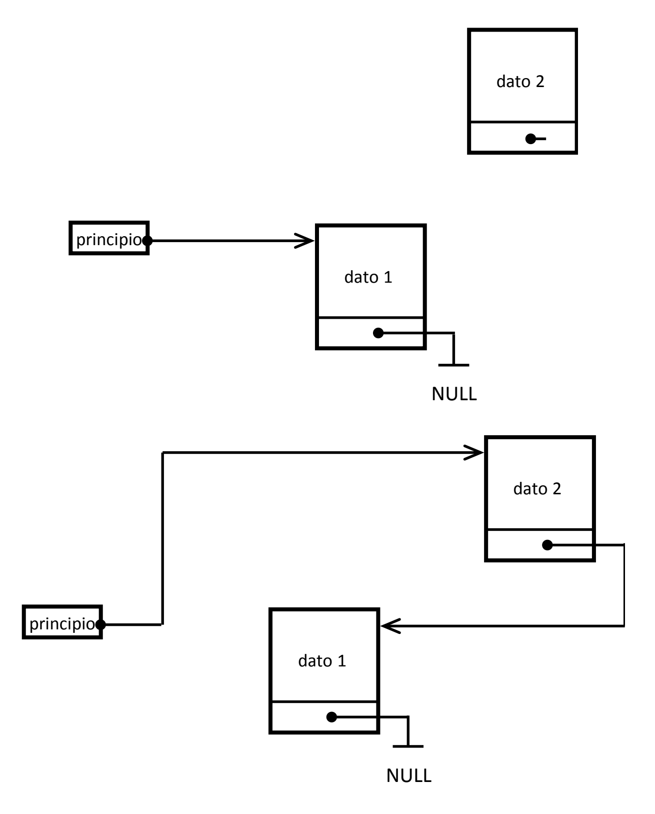
El primer conjunto de datos estaría apuntado por el puntero de inicio, y lo podemos visualizar en el siguiente esquema.



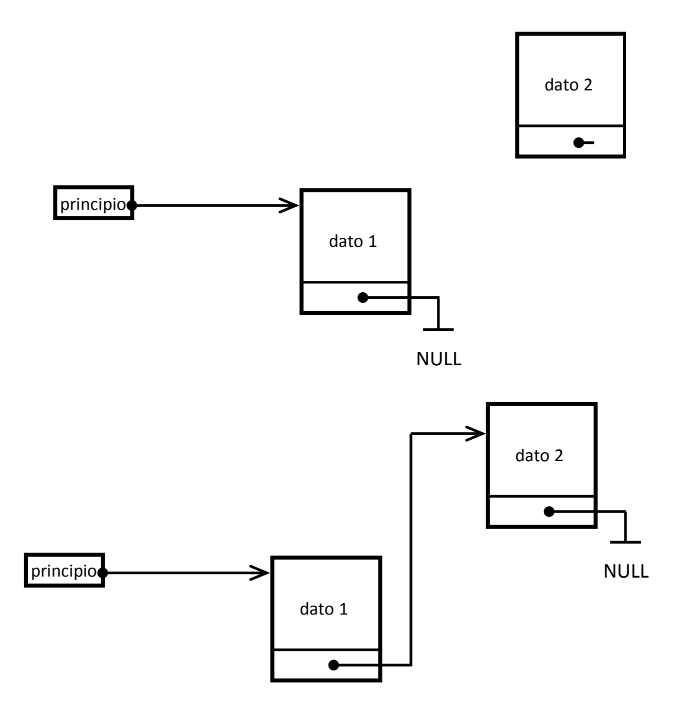
El segundo podría tener dos posibles posiciones para ser insertado:

- Como nuevo primer elemento.
- Como último elemento.

En el primer caso el puntero de inicio debe apuntar al nuevo elemento, el puntero del nuevo elemento debe apuntar al elemento que estaba en la lista y el puntero de este deberá apuntar a NULL.



e la lista entonces el dato que ya pertenecía a ella quedará en la posición actual y apuntará al nuevo conjunto, a su vez el puntero de este nuevo conjunto apuntará a NULL. Este caso es igual al visto anteriormente.



Desde el tercer elemento a insertar, pueden producirse las tres posibilidades que habíamos mencionado inicialmente:

- Como nuevo primer elemento.
- Como elemento intermedio.
- Como último elemento.

Cuando sucede que el nuevo elemento debe ser el primer elemento el indicador de principio deberá apuntar a este nuevo elemento y el indicador del elemento siguiente en el nuevo deberá apuntar al elemento que era el primero.

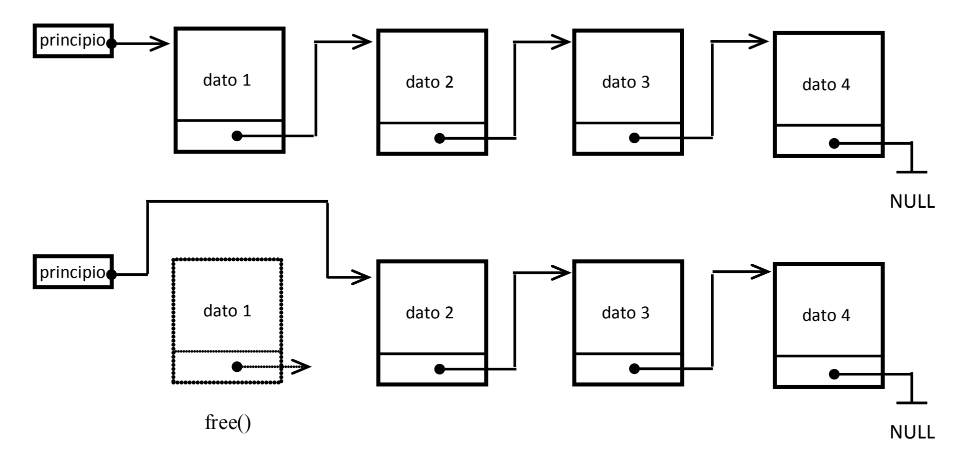
Otra posible ubicación es entre dos elementos existentes, entonces se debe cambiar el enlace del elemento anterior hacia el nuevo y el enlace del nuevo apuntará al posterior.

Por último puede ocurrir que el elemento a ser insertado debe ubicarse al final de la lista, por lo tanto el puntero siguiente del elemento final debe ahora apuntar al nuevo elemento y el puntero siguiente del nuevo debe apuntar a NULL.

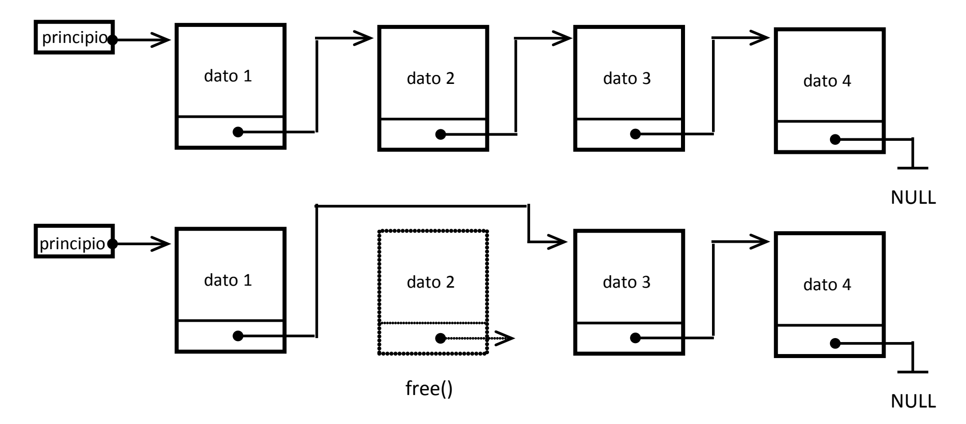
Para leer los elementos de la lista solo se debe comenzar desde la dirección que apunta el indicador de comienzo y luego de presentar los datos, solo se debe actualizar el puntero con el valor del puntero siguiente de la estructura hasta llegar a encontrar el valor NULL.
Cuando se debe buscar un determinado valor en la lista se debe hacer en forma secuencial por lo que hay que comenzar desde le valor que indica el puntero de comienzo hasta encontrar el valor deseado o hasta encontrar el NULL, que indicaría que no se ha encontrado el valor buscado.
Cuando se desee borrar un elemento encontrado en la lista pueden ocurrir tres casos:

- Borrar el primer elemento.
- Borrar un elemento intermedio.
- Borrar el último elemento.

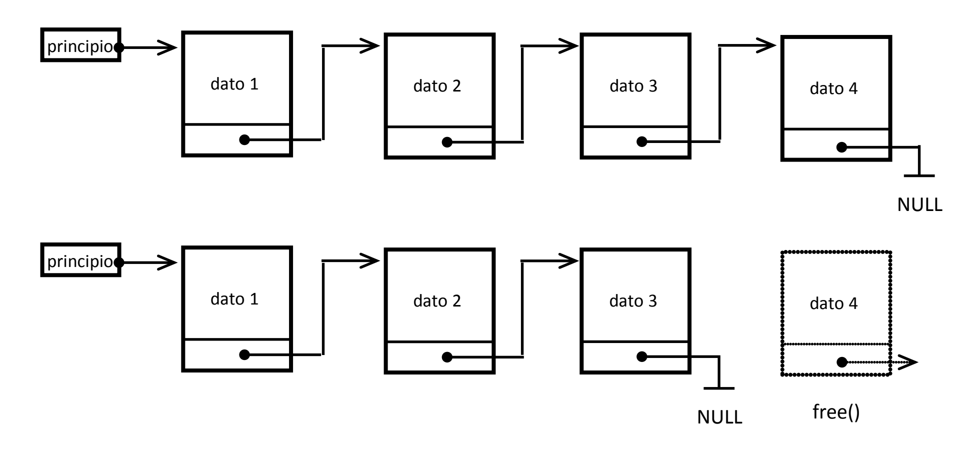
Si es el primer elemento, se debe cambiar el puntero de comienzo a la dirección del segundo elemento. Luego se debe proceder a liberar la memoria del elemento borrado mediante el uso de la función free().



En caso de que sea un elemento intermedio se debe hacer que el puntero siguiente del elemento anterior apunte al elemento siguiente al que será borrado y luego liberar la memoria de este.



Si el elemento a eliminar esta ubicado al final de la lista debemos proceder de la siguiente forma: el puntero siguiente del elemento anterior debe apuntar a NULL y luego liberar la memoria del elemento a borrar.



Ejemplo

```
1.  //---------------------------------------------------------------
2.  #include<stdio.h>
3.  #include<stdlib.h>
4.  #include<conio.h>
```

```
5.  //--------------------------------------------------------------
6.      struct Lis
7.      {
8.          int Dato;
9.          struct Lis *Sig;
10.};
11.//--------------------------------------------------------------
12.void Ingresa(struct Lis **);
13.void Lee(struct Lis **);
14.void Borra(struct Lis **);
15.void Elimina(struct Lis **);
16.//--------------------------------------------------------------
17.int main(void)
18.{
19.  int Opcion;
20.  struct Lis *Inicio=NULL;
21.  for(;;)
22.  {
23.      system("cls");
24.      printf("\n1 - Ingresa datos\n");
25.      printf("2 - Lee datos\n");
26.      printf("3 - Borra datos\n");
27.      printf("4 - Salir\n\n");
28.      printf("Ingrese una opcion ( 1 - 4 ) : ");
29.      Opcion=getche();
30.      switch(Opcion)
31.      {
32.      case '1':
33.          Ingresa(&Inicio);
34.          break;
35.      case '2':
36.          Lee(&Inicio);
37.          break;
38.      case '3':
39.          Borra(&Inicio);
40.          break;
41.      case '4':
42.          Elimina(&Inicio);
43.          exit(0);
44.      }
45.  }
```

```
46.  return 0;
47.}
48.//--------------------------------------------------------------
49.void Ingresa(struct Lis **Principio)
50.{
51.  struct Lis *Actual=*Principio,*Nuevo;
52.  system("cls");
53.  if(!(Nuevo=(struct Lis *)malloc(sizeof(struct Lis))))
54.  {
55.     system("cls");
56.     printf("\n\nNo hay memoria disponible\n");
57.     printf("\n\nPresione una tecla para continuar\n");
58.     getch();
59.     return;
60.  }
61.  printf("\n\nIngrese el dato : ");
62.  scanf("%d",&Nuevo->Dato);
63.  Nuevo->Sig=NULL;
64.  if(!*Principio)
65.  {
66.     *Principio=Nuevo;
67.     return;
68.  }
69.  while(Actual->Sig)
70.     Actual=Actual->Sig;
71.  Actual->Sig=Nuevo;
72.}
73.//--------------------------------------------------------------
74.void Lee(struct Lis **Principio)
75.{
76.  struct Lis *Actual=*Principio;
77.  system("cls");
78.  while(Actual)
79.  {
80.     printf("\nEl dato es : %d",Actual->Dato);
81.     Actual=Actual->Sig;
82.  }
83.  printf("\n\nPresione una tecla para continuar\n");
84.  getch();
85.}
86.//--------------------------------------------------------------
```

```
87.void Borra(struct Lis **Principio)
88.{
89. struct Lis *Actual=*Principio,*Anterior=*Principio;
90. int Valor;
91. system("cls");
92. printf("\nIngrese el dato a borrar : ");
93. scanf("%d",&Valor);
94. while(Actual&&(Valor!=Actual->Dato))
95. {
96.     Anterior=Actual;
97.     Actual=Actual->Sig;
98. }
99. if(Actual&&(*Principio==Actual))
100.        {
101.            *Principio=Actual->Sig;
102.            free(Actual);
103.            return;
104.        }
105.        if(Actual)
106.        {
107.            Anterior->Sig=Actual->Sig;
108.            free(Actual);
109.            return;
110.        }
111.        system("cls");
112.        printf("\nEl dato no ha sido encontrado");
113.        printf("\n\nPresione una tecla para continuar\n");
114.        getch();
115.    }
116.    //---------------------------------------------------------------
117.    void Elimina(struct Lis **Principio)
118.    {
119.        struct Lis *Actual;
120.        while(*Principio)
121.        {
122.            Actual=*Principio;
123.            *Principio=(*Principio)->Sig;
124.            free(Actual);
125.        }
126.    }
127.    //---------------------------------------------------------------
```

En este ejemplo tenemos una aplicación de una lista simplemente enlazada, la que permite ingresar un dato.
El dato está contenido en una estructura llamada Lis que se define entre las líneas 6 a 10. Contiene un puntero a struct Lis llamado Sig que va a ser el encargado de apuntar al elemento siguiente de la lista. Cabe destacar que el tamaño que ocupa un puntero como el mencionado es siempre de 4 bytes, por lo tanto cuando se define la estructura lo que se hace es solamente guardar este espacio para el puntero, el tipo al que apunta se lo coloca para determinar su forma y además para darle la información de cuantos bytes tiene que incrementar o decrementar.
Entre las líneas 17 y 47 se define la función main. En esta se genera un lazo para recorrer un menú de opciones, y se selecciona la función correspondiente mediante una estructura switch-case. Para salir se utiliza la función exit con un parámetro 0 que indica una salida normal del programa.
Desde la línea 49 a 72 se define la función ingresa; su prototipo está ubicado en la línea 12, recibe como parámetro la dirección del puntero al comienzo de la lista (un puntero a puntero a struct Lis llamado Principio) y devuelve void.
Se declarán dos punteros (Actual y Nuevo) a struct Lis que se utilizarán para guardar los valores de los punteros de los elementos de la lista. A Nuevo se lo utilizará para apuntar al nuevo elemento. A Actual, para que apunte al elemento Actual en la lista.
En la línea 53 se solicita memoria para ubicar al nuevo elemento mediante la función malloc. Esta devuelve el valor del bloque al puntero Nuevo y se pregunta por el resultado del pedido en caso de que hubiese devuelto un NULL se informa el hecho y luego se retorna a main (líneas 55 a 59).
En el caso de que el resultado del requerimiento de memoria fuese satisfactorio se ingresará el nuevo dato (línea 62).
En la línea 63 se le asigna al puntero Nuevo->Sig el valor NULL ya que va a ser el último elemento de la lista.
En la línea 64 se pregunta por el valor del contenido del puntero a puntero Principio. En caso de que este sea NULL, indica que la lista está vacía y por lo tanto se le asigna a este el valor de Nuevo y termina la función retornando a main (líneas 66 y 67).
Si no fuese así, quiere decir que hay por lo menos un elemento en la lista y por lo tanto hay que proceder a buscar el lugar en donde se lo agregará. Esto se realiza entre las líneas 69 y 70. Como primer paso, se asignará al puntero Actual el valor del contenido del puntero a puntero Principio (línea 51). Ente las líneas 69 y 70 se ejecuta un lazo del tipo while con la condición (Actual->Sig), que verifica que el puntero Actual->Sig no sea NULL; dentro del lazo se Actualiza el valor del puntero Actual. A Actual se le asigna el valor de Actual->Sig, es decir el valor del elemento siguiente. De tal manera se recorrerá la lista hasta encontrar el fin de la lista para agregar el nuevo elemento.
Al llegar al final el valor de Actual->Sig será NULL y por lo tanto el nuevo elemento se va a ubicar allí. Esto se realiza asignándole al puntero Actual->Sig el valor del puntero Nuevo (línea 71), finalizando la ejecución de la función.
La función *lee* se define entre las líneas 74 y 85; esta recibe como parámetro la dirección del puntero a inicio de la lista en un puntero a puntero a struct Lis llamado Principio y devuelve void. El prototipo está en la línea 13.
Se define un puntero Actual al que se lo inicializa con el contenido de Principio (línea 76). Este puntero servirá para recorrer la lista. La lista se recorre mediante un lazo while con la condición de que el puntero Actual sea distinto de NULL (líneas 78 a 82). Dentro del lazo se imprimen los valores (línea 80), en la línea siguiente se modifica el valor de Actual a Actual->Sig para ir al elemento siguiente en la lista. Al finalizar este recorrido se retornara a main.
En las líneas 87 a 115 se define la función borra; su prototipo está en la línea 14. En ella se declarán dos punteros Actual y Anterior, inicializándolos con la dirección de inicio de la lista. Estos se van a utilizar para recorrer la lista.
En las líneas 92 y 93 se ingresa el valor que se quiere eliminar. Luego se comienza a recorrer la lista para encontrarlo mediante un lazo. Los punteros Actual y Anterior están inicializados con el contenido de Principio, después de esto se ingresa a un ciclo while con la condición

Actual&&(Valor!=Actual->Dato). Se recorrerá la lista hasta encontrar el valor deseado o que el puntero Actual sea distinto de NULL; en el cuerpo del while se modifican los valores de los punteros Actual y Anterior; al primero se le asigna el valor Actual->Sig para poder recorrer la lista y al segundo el valor de Actual para no perder el lugar del elemento anterior (líneas 94 a 98). Si se encuentra que el valor del puntero Actual no es NULL, quiere decir que se ha encontrado el valor; además, si el contenido de Principio es igual al de Actual, quiere decir que el elemento es el primero. Se procede a modificar el contenido de Principio con el puntero Actual->Sig y luego se libera la memoria utilizando free(), retornando a main (líneas 99 a 104). Si ocurre que Actual es distinto del contenido de Principio, significa que es un elemento distinto al primero actualizando el puntero Anterior->Sig con el valor de Actual->Sig y luego se libera la memoria con la función free, retornando a main (líneas 105 y 110).
Desde la línea 111 a 114 se indica que el dato deseado no se ha encontrado.
Desde la línea 117 a 126 se encuentra definida la función Elimina que se utiliza para eliminar los elementos que quedaron en la lista para devolver al sistema operativo los bloque de memoria solicitados antes de la finalización del programa. Su prototipo se encuentra en la línea 15. Esta función recorre la lista modificando el puntero Principio y liberando la memoria mediante el uso del puntero Actual.

```
1.    void Ingresa(struct Lis **Principio)
2.    {
3.      struct Lis *Actual,*Nuevo,*Anterior;
4.      if(!(Nuevo=(struct Lis *)malloc(sizeof(struct Lis))))
5.      {
6.        printf("\n\nNo hay memoria disponible\n");
7.        printf("\n\nPresione una tecla para continuar\n");
8.        getch();
9.        return;
10.   }
11.   printf("\n\nIngrese el dato : ");
12.   scanf("%d",&Nuevo->Dato);
13.   if(!*Principio)
14.   {
15.     *Principio=Nuevo;
16.     Nuevo->Sig=NULL;
17.     return;
18.   }
19.   Anterior=*Principio;
20.   Actual=*Principio;
21.   while(Actual&&(Actual->Dato>Nuevo->Dato))
22.   {
23.     Anterior=Actual;
24.     Actual=Actual->Sig;
25.   }
26.   if(!Actual)
27.   {
28.     Anterior->Sig=Nuevo;
29.     Nuevo->Sig=NULL;
30.     return;
31.   }
32.   if(Anterior==Actual)
33.   {
34.     *Principio=Nuevo;
```

```
35.     Nuevo->Sig=Anterior;
36.     return;
37.   }
38.   Anterior->Sig=Nuevo;
39.   Nuevo->Sig=Actual;
40. }
```

La función ingresa escrita difiere de la función ingresa que está en el programa anterior en que en esta ultima cada nuevo elemento se inserta en la lista en la ubicación correspondiente a fin de que la lista siempre quede ordenada.
Ello se hace entre las líneas 13 a 39. En la 13 se verifica si el contenido del puntero Principo es NULL, esto significa que la lista esta vacía. En la línea 15 se apunta el inicio de la lista al elemento Nuevo, en la 16 se le asigna al puntero Nuevo->Sig el valor NULL para indicar que es el ultimo elemento y en la 17 se retorna a la función main.
En las líneas 19y 20 se inicializan los punteros Anterior y Actual con el inicio de la lista, para comenzar a recorrerla y encontrar la posición en donde se ubicara el nuevo elemento.
Entre las líneas 21 y 25 se comienza a recorrer la lista para poder encontrar la posición en donde se insertará el nuevo elemento. Esto se realiza mediante un lazo while con la condición Actual&&(Actual->Dato>Nuevo->Dato). Esta condición hace que el lazo finalice cuando se encuentre el fin de la lista o se haya encontrado que el elemento al que apunta Actual contenga un dato mayor que el que se quiere insertar. En las líneas 23 y 24 se actualizan los valores de los punteros Anterior y Actual. A Anterior se le asigna el valor que tiene el puntero Actual de tal forma que este apunte al elemento anterior para que con este se pueda enlazar el nuevo elemento. Al puntero Actual se le asigna la dirección del elemento siguiente para poder encontrar el fin de lista o la ubicación del elemento que apuntará el nuevo elemento para poder enlazar a la lista.
Entre las líneas 26 a 31 se verifica que se llegó al fin de la lista. Esta situación se obtiene mediante la evaluación del puntero Actual. Si este tiene el valor NULL, indica que se llegó al fin de la lista. En esta situación se le asigna a Anterior->Sig la dirección del nuevo elemento (línea 28). Y como el nuevo elemento es el último, a su puntero Sig se le asigna el valor NULL para indicar el fin de la lista (línea 29) retornando a la función main.
Desde la línea 32 a 37 se verifica que el elemento se insertará en el primer lugar de la lista. Esta condición se obtiene analizando el estado de los punteros Anterior y Actual. Si estos son iguales indica que no se ha comenzado a recorrer la lista, por lo que el nuevo elemento se insertará en el primer lugar. Esto se realiza asignándole al contenido de Principio la dirección del nuevo elemento (línea 34). Para enlazar la lista a Nuevo->Sig se le asigna el valor del puntero Anterior que contiene la ubicación del elemento que era el inicio de la lista (línea 35) y se retorna a main.
Entre las líneas 38 y 39 se ubica al elemento en una posición intermedia en la lista asignándole al puntero Anterior->Sig la dirección de nuevo y a Nuevo->Sig la dirección del puntero Actual enlazando la lista.

ÍNDICE



MATERIAL DE DISTRIBUCIÓN GRATUITA

Universidad Nacional de Moreno
Av. Bartolomé Mitre N° 1891, Moreno (B1744OHC), prov. de Buenos Aires, Argentina
(+54 237) 466-7186/1529/4530
(+54 237) 462-8629
(+54 237) 460-1309
www.unm.edu.ar
www.facebook.com/unimoreno

